Unsupervised Machine Learning

Teen Market Segmentation Using K-means Clustering

Interacting with friends on a social networking service (SNS) has become a rite of passage for teenagers around the world.

The many millions of teenage consumers using such sites have attracted the attention of marketers struggling to find an edge in an increasingly competitive market.

One way to gain this edge is to identify segments of teenagers who share similar tastes, so that clients can avoid targeting advertisements to teens with no interest in the product being sold.

For instance, sporting apparel is likely to be a difficult sell to teens with no interest in sports.

Dataset Information

The dataset represents a random sample of 30,000 U.S. high school students who had profiles on a well-known SNS in 2006.

To protect the users’ anonymity, the SNS will remain unnamed. The data was sampled evenly across four high school graduation years (2006 through 2009) representing the senior, junior, sophomore, and freshman classes at the time of data collection.

The dataset contatins 40 variables like: gender, age, friends, basketball, football, soccer, softball, volleyball,swimming, cute, sexy, kissed, sports, rock, god, church, bible, hair, mall, clothes, hollister, drugs etc whcih shows their interests.

The final dataset indicates, for each person, how many times each word appeared in the person’s SNS profile.

Load Libraries

In [1]:
```
# Importing Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```
pd.set_option('display.max_columns',None)
data = pd.read_csv("G:/TCS Study/TCS Git Hub Projects/Unsupervised Learning Project/sns
data.head()
```

Out[2]:

	gradyear	gender	age	friends	basketball	football	soccer	softball	volleyball	swimming	chee
0	2006	M	18.982	7	0	0	0	0	0	0	
1	2006	F	18.801	0	0	1	0	0	0	0	
2	2006	M	18.335	69	0	1	0	0	0	0	
3	2006	F	18.875	0	0	0	0	0	0	0	
4	2006	NaN	18.995	10	0	0	0	0	0	0	

Summary Statistics

Summary Statistics of Numerical Variables

In [3]:

```
data.describe()
```

Out[3]:

	gradyear	age	friends	basketball	football	soccer	
count	30000.000000	24914.000000	30000.000000	30000.000000	30000.000000	30000.000000	3000(
mean	2007.500000	17.993950	30.179467	0.267333	0.252300	0.222767	(
std	1.118053	7.858054	36.530877	0.804708	0.705357	0.917226	(
min	2006.000000	3.086000	0.000000	0.000000	0.000000	0.000000	(
25%	2006.750000	16.312000	3.000000	0.000000	0.000000	0.000000	(
50%	2007.500000	17.287000	20.000000	0.000000	0.000000	0.000000	(
75%	2008.250000	18.259000	44.000000	0.000000	0.000000	0.000000	(
max	2009.000000	106.927000	830.000000	24.000000	15.000000	27.000000	1

Summary Statistics of Categorical Variables

In [4]:

```
data.describe(include='object')
```

Out[4]:

	gender
count	27276
unique	2
top	F
freq	22054

Treating Missing Values

In [5]:

```
data.isnull().sum()
```

Out[5]:
```
gradyear           0
gender          2724
age             5086
friends            0
basketball         0
football           0
soccer             0
softball           0
volleyball         0
swimming           0
cheerleading       0
baseball           0
tennis             0
sports             0
cute               0
sex                0
sexy               0
hot                0
kissed             0
dance              0
band               0
marching           0
music              0
rock               0
god                0
church             0
jesus              0
bible              0
hair               0
dress              0
blonde             0
mall               0
shopping           0
clothes            0
hollister          0
abercrombie        0
die                0
death              0
drunk              0
drugs              0
dtype: int64
```

A total of 5,086 records have missing ages. Also concerning is the fact that the minimum and maximum values seem to be unreasonable; it is unlikely that a 3 year old or a 106 year old is attending high school.

Let's have a look at the number of male and female candidates in our dataset.

In [6]:
```
data['gender'].value_counts()
```

Out[6]:
```
F    22054
M     5222
Name: gender, dtype: int64
```

Let's have a look at the number of male, female and msiing values.

In [7]:
```
data['gender'].value_counts(dropna = False)
```

```
F      22054
```

Out[7]:
```
M       5222
NaN     2724
Name: gender, dtype: int64
```

There are 22054 female, 5222 male teen students and 2724 missing values.

Now we are going to fill all the null values in gender column with “No Gender”.

In [12]:
```
data['gender'].fillna('not disclosed', inplace = True)
```

In [14]:
```
data['gender'].isnull().sum()
```

Out[14]:
```
0
```

Also, the age cloumn has 5086 missing values.

One way to deal with these missing values would be to fill the missing values with the average age of each graduation year.

In [15]:
```
data.groupby('gradyear')['age'].mean()
```

Out[15]:
```
gradyear
2006    19.137241
2007    18.391459
2008    17.523867
2009    16.876025
Name: age, dtype: float64
```

From the above summary we can observe that the mean age differs by roughly one year per change in graduation year. This is not at all surprising, but a helpful finding for confirming our data is reasonable.

We now fill the missing values for each graduation year with the mean that we got as above.

In [16]:
```
data['age'] = data.groupby('gradyear').transform(lambda x : x.fillna(x.mean()))
```

In [17]:
```
data['age'].isnull().sum()
```

Out[17]:
```
0
```

We don't have any missing values in the 'age' column.

In [18]:
```
data.isnull().sum()
```

Out[18]:
```
gradyear        0
gender          0
age             0
friends         0
basketball      0
football        0
soccer          0
softball        0
volleyball      0
```

```
swimming       0
cheerleading   0
baseball       0
tennis         0
sports         0
cute           0
sex            0
sexy           0
hot            0
kissed         0
dance          0
band           0
marching       0
music          0
rock           0
god            0
church         0
jesus          0
bible          0
hair           0
dress          0
blonde         0
mall           0
shopping       0
clothes        0
hollister      0
abercrombie    0
die            0
death          0
drunk          0
drugs          0
dtype: int64
```

From the above summary we can see that there are no missing values in the dataset.

Treating Outliers

The original age range contains value from 3 - 106, which is unrealistic because student at age of 3 or 106 would not attend high school.

A reasonable age range for people attending high school will be the age range between 13 to 21.

The rest should be treated as outliers keeping the age of student going to high school in mind. Let's detect the outliers using a box plot below.

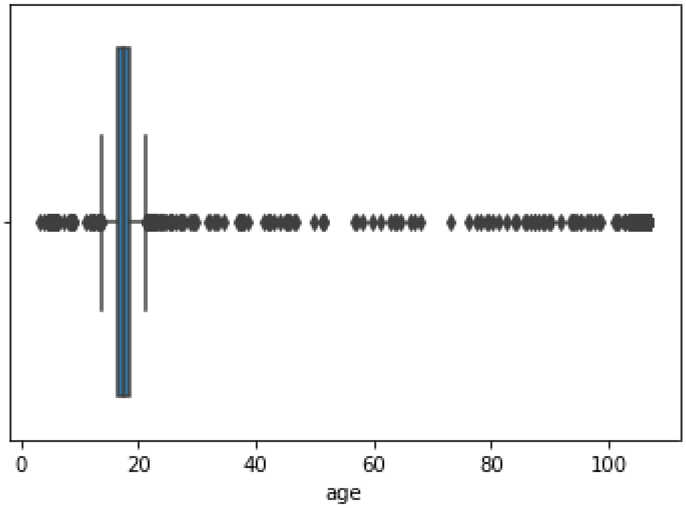
In [19]:

```
sns.boxplot(data['age'])
```

```
C:\Anaconda\envs\ml\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[19]:

```
<AxesSubplot:xlabel='age'>
```



In [22]:
```
q1 = data['age'].quantile(0.25)
q3 = data['age'].quantile(0.75)
iqr = q3-q1
print(iqr)
```

```
1.8874592240696728
```

In [23]:
```
df = data[(data['age'] > (q1 - 1.5*iqr)) & (data['age'] < (q3 + 1.5*iqr))]
```

In [24]:
```
df['age'].describe()
```

Out[24]:
```
count    29633.000000
mean        17.377469
std          1.147764
min         13.719000
25%         16.501000
50%         17.426000
75%         18.387000
max         21.158000
Name: age, dtype: float64
```

From the above summary we can observe that after treating the outliers the mininmum age is 13.719000 and the maximum age is 21.158000

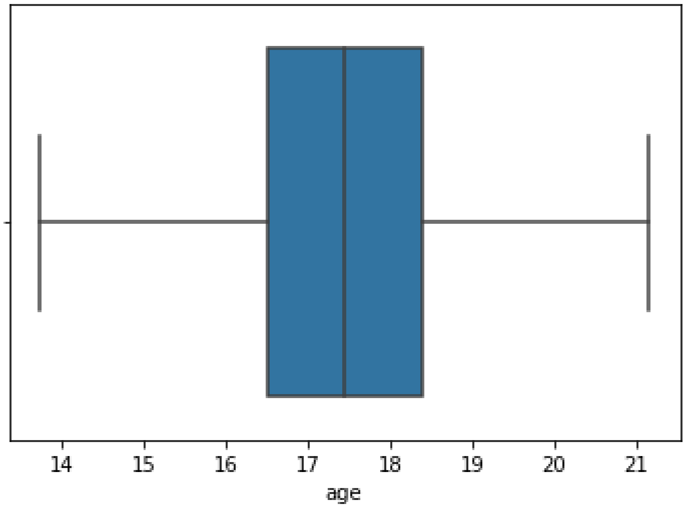
In [25]:
```
df.shape
```

Out[25]:
```
(29633, 40)
```

In [26]:
```
sns.boxplot(df['age'])
```

```
C:\Anaconda\envs\ml\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the
following variable as a keyword arg: x. From version 0.12, the only valid positional arg
ument will be `data`, and passing other arguments without an explicit keyword will resul
t in an error or misinterpretation.
  FutureWarning
```

Out[26]:
```
<AxesSubplot:xlabel='age'>
```



From the above boxplot we observe that there are no outliers in the age column

Data Preprocessing

A common practice employed prior to any analysis using distance calculations is to normalize or z-score standardize the features so that each utilizes the same range.

By doing so, we can avoid a problem in which some features come to dominate solely because they have a larger range of values than the others.

The process of z-score standardization rescales features so that they have a mean of zero and a standard deviation of one.

This transformation changes the interpretation of the data in a way that may be useful here.

Specifically, if someone mentions Swimming three times on their profile, without additional information, we have no idea whether this implies they like Swimming more or less than their peers.

On the other hand, if the z-score is three, we know that that they mentioned Swimming many more times than the average teenager.

In [27]:

```
names = df.columns[5:40]
scaled_feature = data.copy()
names
```

Out[27]:

```
Index(['football', 'soccer', 'softball', 'volleyball', 'swimming',
       'cheerleading', 'baseball', 'tennis', 'sports', 'cute', 'sex', 'sexy',
       'hot', 'kissed', 'dance', 'band', 'marching', 'music', 'rock', 'god',
       'church', 'jesus', 'bible', 'hair', 'dress', 'blonde', 'mall',
       'shopping', 'clothes', 'hollister', 'abercrombie', 'die', 'death',
       'drunk', 'drugs'],
      dtype='object')
```

In [28]:

```
features = scaled_feature[names]
```

In [29]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(features.values)
```

In [30]:
```
features = scaler.transform(features.values)
```

In [31]:
```
scaled_feature[names] = features
scaled_feature.head()
```

Out[31]:

	gradyear	gender	age	friends	basketball	football	soccer	softball	volleyball	swimn
0	2006	M	18.982	7	0	-0.357697	-0.242874	-0.217928	-0.22367	-0.259
1	2006	F	18.801	0	0	1.060049	-0.242874	-0.217928	-0.22367	-0.259
2	2006	M	18.335	69	0	1.060049	-0.242874	-0.217928	-0.22367	-0.259
3	2006	F	18.875	0	0	-0.357697	-0.242874	-0.217928	-0.22367	-0.259
4	2006	not disclosed	18.995	10	0	-0.357697	-0.242874	-0.217928	-0.22367	-0.259

Convert Object Variable to Numeric

In [32]:
```
def gender_to_numeric(x):
    if x=='M':
        return 1
    if x=='F':
        return 2
    if x=='not disclosed':
        return 3
```

In [33]:
```
scaled_feature['gender'] = scaled_feature['gender'].apply(gender_to_numeric)
scaled_feature['gender'].head()
```

Out[33]:
```
0    1
1    2
2    1
3    2
4    3
Name: gender, dtype: int64
```

Checking the transformed values

In [34]:
```
scaled_feature.head()
```

Out[34]:

	gradyear	gender	age	friends	basketball	football	soccer	softball	volleyball	swimmir
0	2006	1	18.982	7	0	-0.357697	-0.242874	-0.217928	-0.22367	-0.2599
1	2006	2	18.801	0	0	1.060049	-0.242874	-0.217928	-0.22367	-0.2599
2	2006	1	18.335	69	0	1.060049	-0.242874	-0.217928	-0.22367	-0.2599

	gradyear	gender	age	friends	basketball	football	soccer	softball	volleyball	swimmir
3	2006	2	18.875	0	0	-0.357697	-0.242874	-0.217928	-0.22367	-0.25997
4	2006	3	18.995	10	0	-0.357697	-0.242874	-0.217928	-0.22367	-0.25997

Model Building

Building the K-means Model

In [35]:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, random_state=0, n_jobs=-1)
```

In [36]:

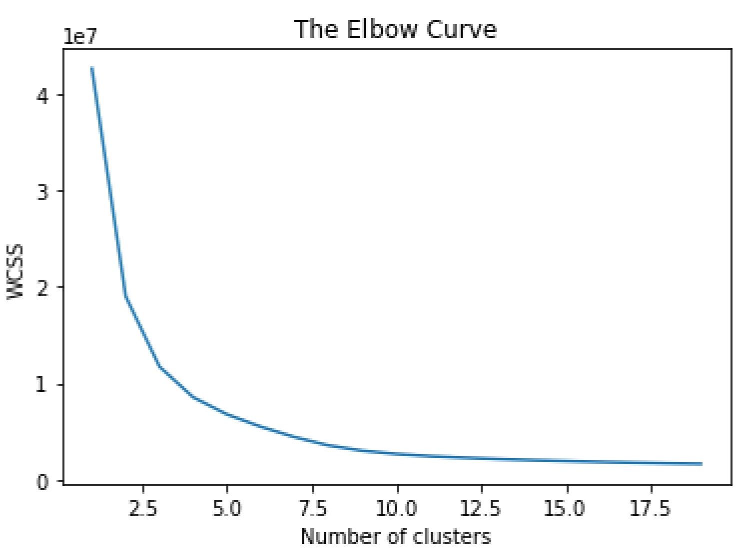
```
model = kmeans.fit(scaled_feature)
```

```
C:\Anaconda\envs\ml\lib\site-packages\sklearn\cluster\_kmeans.py:793: FutureWarning: 'n_
jobs' was deprecated in version 0.23 and will be removed in 1.0 (renaming of 0.25).
  " removed in 1.0 (renaming of 0.25).", FutureWarning)
```

Elbow Method

In [37]:

```
# Creating a funtion with KMeans to plot "The Elbow Curve"
wcss = []
for i in range(1,20):
    kmeans = KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0
    kmeans.fit(scaled_feature)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,20),wcss)
plt.title('The Elbow Curve')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') ##WCSS stands for total within-cluster sum of square
plt.show()
```



The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

Our Elbow point is around cluster size of 5. We will use k=5 to further interpret our clustering result.

Fit K-Means clustering for k=5

In [38]:

```
kmeans = KMeans(n_clusters=5)
kmeans.fit(scaled_feature)
```

Out[38]:

```
KMeans(n_clusters=5)
```

As a result of clustering, we have the clustering label. Let's put these labels back into the original numeric data frame.

In [39]:

```
len(kmeans.labels_)
```

Out[39]:

```
30000
```

In [41]:

```
data['cluster'] = kmeans.labels_
data.head()
```

Out[41]:

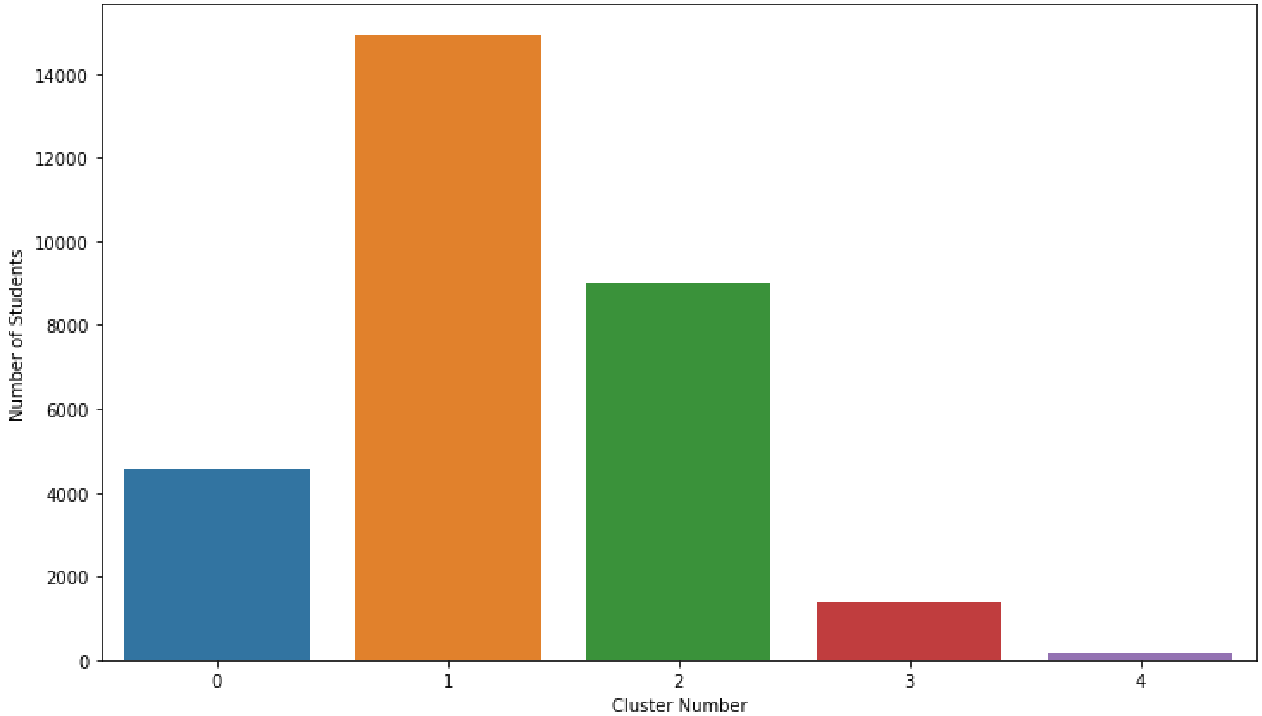
	gradyear	gender	age	friends	basketball	football	soccer	softball	volleyball	swimming	ch
0	2006	M	18.982	7	0	0	0	0	0	0	
1	2006	F	18.801	0	0	1	0	0	0	0	
2	2006	M	18.335	69	0	1	0	0	0	0	
3	2006	F	18.875	0	0	0	0	0	0	0	
4	2006	not disclosed	18.995	10	0	0	0	0	0	0	

Interpreting Clustering Results

Let's see cluster sizes first

In [42]:

```
plt.figure(figsize=(12,7))
axis = sns.barplot(x=np.arange(0,5,1),y=data.groupby(['cluster']).count()['age'].values
x=axis.set_xlabel("Cluster Number")
x=axis.set_ylabel("Number of Students")
```



From the above plot we can see that cluster 0 is the largest and cluster 1 has fewest teen students

Let' see the number of students belonging to each cluster

In [43]:
```
size_array = list(data.groupby(['cluster']).count()['age'].values)
size_array
```

Out[43]:
```
[4541, 14915, 8992, 1378, 174]
```

let's check the cluster statistics

In [45]:
```
data.groupby(['cluster']).mean()[['basketball', 'football','soccer', 'softball','volley
```

Out[45]:

cluster	basketball	football	soccer	softball	volleyball	swimming	cheerleading	baseball	te
0	0.333187	0.284519	0.269324	0.244440	0.183660	0.159216	0.136093	0.127505	0.095
1	0.223332	0.228763	0.192021	0.121086	0.109420	0.115924	0.086021	0.090714	0.080
2	0.289257	0.267682	0.245440	0.170151	0.169484	0.149244	0.110098	0.112211	0.093
3	0.376633	0.301161	0.248186	0.255443	0.192308	0.145864	0.201742	0.133527	0.089
4	0.321839	0.247126	0.270115	0.218391	0.224138	0.212644	0.172414	0.132184	0.086

The cluster center values shows each of the cluster centroids of the coordinates.

The row referes to the five clusters,the numbers across each row indicates the cluster's average value for the interest listed at the top of the column.

Positive values are above the overall mean level.

In [ ]:

```

```