In [1]:
```
import os
import sys
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")

%matplotlib inline

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Load Data

In [2]:
```
df_baskets = pd.read_csv('./retail_baskets.csv')
df_customers = pd.read_csv('./retail_customers.csv')
df_transactions = pd.read_csv('./retail_transactions.csv',encoding='latin-1')

x = df_transactions.drop(['InvoiceNo','CustomerID','InvoiceDate'], axis=1)
```

Filter out unique data points

In [3]:
```
col_obj = x.select_dtypes(include='object')
col_obj = col_obj.loc[:, col_obj.nunique() < 5000]

x.dtypes
col_obj.nunique()
col_cat = list(col_obj.columns)
col_cat
```

Out[3]:
```
StockCode       object
Description     object
Quantity         int64
UnitPrice      float64
Country         object
dtype: object
```

Out[3]:
```
StockCode      4070
Description    4223
Country          38
dtype: int64
```

Out[3]:
```
['StockCode', 'Description', 'Country']
```

Encode string data for grouping

In [4]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
xle = x.copy()

for col in col_cat:
    xle[col] = le.fit_transform(x[col].astype(str))
```

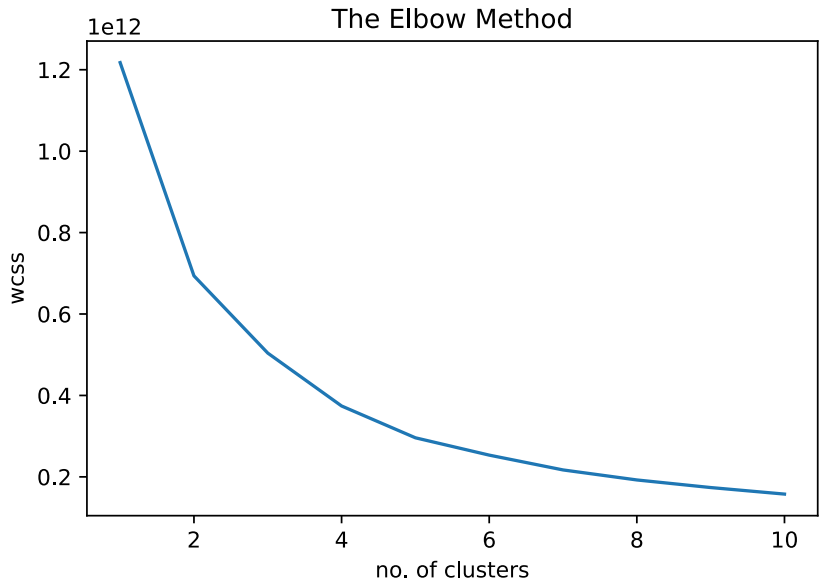
Determine the number of clusters to use

Here I will use the "Elbow Method" for determining the optimum cluster number for the KMeans++ algorithm. The last visible "Elbow" joint will provide the optimal number.

In [5]:
```
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0).fit(xle)
    wcss.append(kmeans.inertia_)

# Code to visualize the elbow
plt.plot(range(1,11), wcss)
_ = plt.title('The Elbow Method')
_ = plt.xlabel('no. of clusters')
_ = plt.ylabel('wcss')
plt.show()
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x7efb9df3b1f0>]
```



Build model

In [6]:
```
kmeansmodel = KMeans(n_clusters=2, init='k-means++', random_state=0)
y_kmeans = kmeansmodel.fit_predict(xle)
```

Unsupervised learning visualization

This visualization will show the key groups that line up based on transaction data.

In [7]:
```
X = np.array(xle)
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label
= 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', labe
l = 'Cluster 2')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 3
00, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.legend()
plt.show()
```

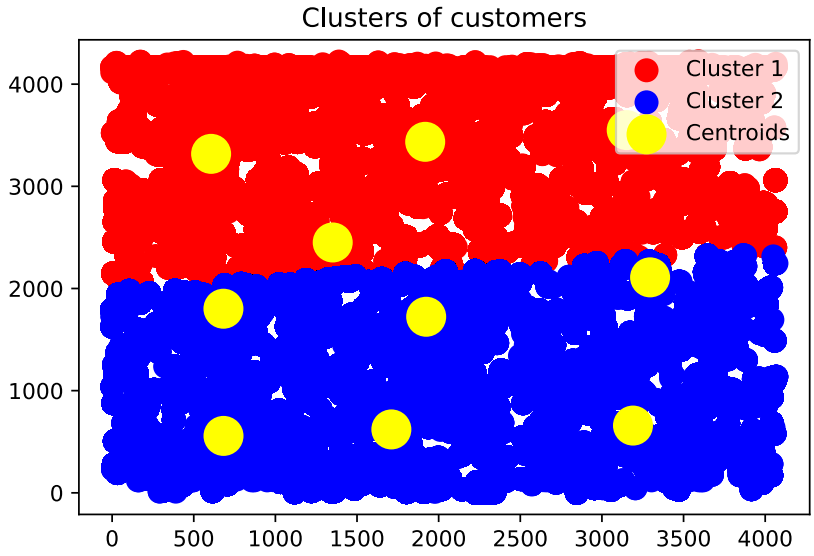
Out[7]: <matplotlib.collections.PathCollection at 0x7efb9de704f0>

Out[7]: <matplotlib.collections.PathCollection at 0x7efb9de70400>

Out[7]: <matplotlib.collections.PathCollection at 0x7efb9de70e50>

Out[7]: Text(0.5, 1.0, 'Clusters of customers')

Out[7]: <matplotlib.legend.Legend at 0x7efba1a06040>



Insight

Each of the centroids represents the concentration of each column in the data set clustered by 2 segmentation groups determined by KMeans. To utilize this data, a company can target each type of customer based on the the different goups laid out by KMeans.