Market segmentation

Import the relevant libraries

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
```

Load the data

In [4]:
```
# Load the data
data = pd.read_csv ('Example.csv')
```

In [5]:
```
# Check what's inside
data
```

Out[5]:

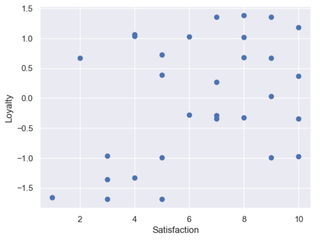
	Satisfaction	Loyalty
0	4	-1.33
1	6	-0.28
2	5	-0.99
3	7	-0.29
4	4	1.06
5	1	-1.66
6	10	-0.97
7	8	-0.32
8	8	1.02
9	8	0.68
10	10	-0.34
11	5	0.39
12	5	-1.69
13	2	0.67
14	7	0.27
15	9	1.36
16	8	1.38
17	7	1.36
18	7	-0.34
19	9	0.67
20	10	1.18
21	3	-1.69
22	4	1.04
23	3	-0.96
24	6	1.03
25	9	-0.99
26	10	0.37
27	9	0.03
28	3	-1.36
29	5	0.73

Plot the data

Create a preliminary plot to see if you can spot something

In [6]:
```
# We are creating a scatter plot of the two variables
plt.scatter(data['Satisfaction'],data['Loyalty'])
# Name your axes
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
```

Out[6]:
```
Text(0, 0.5, 'Loyalty')
```



Select the features

In [7]:
```
# Select both features by creating a copy of the data variable
x = data.copy()
```

Clustering

In [8]:
```
kmeans = KMeans(2)
kmeans.fit(x)
```

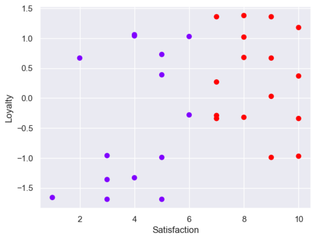
Out[8]:
```
KMeans(n_clusters=2)
```

Clustering results

In [9]:
```
clusters = x.copy()
clusters['cluster_pred']=kmeans.fit_predict(x)
```

In [10]:
```
plt.scatter(clusters['Satisfaction'],clusters['Loyalty'],c=clusters['cluster_pred'],cmap='rainbow')
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
```

Out[10]:
```
Text(0, 0.5, 'Loyalty')
```



Standardize the variables

Let's standardize and check the new result

In [11]:
```
from sklearn import preprocessing
# Scale the inputs
# preprocessing.scale scales each variable (column in x) with respect to itself
# The new result is an array
x_scaled = preprocessing.scale(x)
x_scaled
```

Out[11]:
```
array([[-0.93138063, -1.3318111 ],
       [-0.15523011, -0.28117124],
       [-0.54330537, -0.99160391],
       [ 0.23284516, -0.29117733],
       [-0.93138063,  1.059645  ],
       [-2.09560642, -1.6620122 ],
       [ 1.39707095, -0.97159172],
       [ 0.62092042, -0.32119561],
       [ 0.62092042,  1.01962097],
       [ 0.62092042,  0.67941378],
       [ 1.39707095, -0.3412078 ],
       [-0.54330537,  0.38923705],
       [-0.54330537, -1.69203048],
       [-1.70753116,  0.669407  ],
       [ 0.23284516,  0.26916393],
       [ 1.00899568,  1.35982816],
       [ 0.62092042,  1.37984035],
       [ 0.23284516,  1.35982816],
       [ 0.23284516, -0.3412078 ],
       [ 1.00899568,  0.669407  ],
       [ 1.39707095,  1.17971847],
       [-1.31945589, -1.69203048],
       [-0.93138063,  1.03963316],
       [-1.31945589, -0.96158562],
       [-0.15523011,  1.02962706],
       [ 1.00899568, -0.99160391],
       [ 1.39707095,  0.369224  ],
       [ 1.00899568,  0.029017  ],
       [-1.31945589, -1.36182938],
       [-0.54330537,  0.72944425]])
```

Take advantage of the Elbow method

In [12]:
```
wcss =[]

# Create all possible cluster solutions with a loop
# We have chosen to get solutions from 1 to 9 clusters; you can ammend that if you wish
for i in range(1,10):
    # Cluster solution with i clusters
    kmeans = KMeans(i)
    # Fit the STANDARDIZED data
    kmeans.fit(x_scaled)
    # Append the WCSS for the iteration
    wcss.append(kmeans.inertia_)

# Check the result
wcss
```

```
C:\Users\agawr\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to hav
e a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[12]:
```
[60.0,
 29.818973024723147,
 17.913349527387965,
 10.247181805928 42,
 7.7926951539371 87,
 6.5498367915599 33,
 5.3266311247539 26,
 4.3371107502370 59,
 3.8538053142602 88]
```

In [13]:
```
# Plot the number of clusters vs WCSS
plt.plot(range(1,10),wcss)
# Name your axes
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

Out[13]:
```
Text(0, 0.5, 'WCSS')
```