K- Nearest Neighbors on Customer Segmentation

K-Nearest Neighbors is a supervised machine learning algorithm. The data is 'trained' with data points corresponding to their classification. Once a point is to be predicted,It considers the 'K' Nearest Neighbors (points) when it predicts the classification of the test point.

Imagine a telecommunications provider has segmented its customer base by service usage patterns, categorizing the customers into four groups. The company can customize offers for individual customers. Given the dataset, with predefined labels, we can build a model that can be used to predict class of a new or unknown case.

The example focuses on using demographic data, such as region, age, and marital, to predict usage patterns.

The target field, called **custcat**, has four possible values that correspond to the four customer groups, as follows: 1- Basic Service 2- E-Service 3- Plus Service 4- Total Service

Our objective is to build a classifier, to predict the class of unknown cases. We will use a specific type of classification called K nearest neighbour.

In [173]:

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

In [174]:

```
data = pd.read_csv("data/customer-segmentation.csv")
```

In [175]:

```
print(data.shape)
```

```
(1000, 12)
```

In [176]:

```
#Print no of integers, floats and strings
data.dtypes.value_counts()
```

Out[176]:

```
int64      10
float64     2
dtype: int64
```

In [177]:

```
data.head()
```

Out[177]:

	region	tenure	age	marital	address	income	ed	employ	retire	gender
0	2	13	44	1	9	64.0	4	5	0.0	0
1	3	11	33	1	7	136.0	5	5	0.0	0
2	3	68	52	1	24	116.0	1	29	0.0	1
3	2	33	33	0	12	33.0	2	0	0.0	1
4	2	23	30	1	9	30.0	1	2	0.0	0

In [178]:

```
#Check how balanced our data set is?
data['custcat'].value_counts()
```

Out[178]:

```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

```
Its fairly a balanced set.
```

Preprocessing Steps

1. Select Features .
2. Split the data into train and test sets.

1. Select Features and Normalize data.

In [179]:

```
X = data[['region', 'tenure','age', 'marital', 'address', 'income', 'ed
X[0:5]
```

Out[179]:

```
array([[  2.,  13.,  44.,   1.,   9.,  64.,   4.,   5.,
0.,   0.,   2.],
       [  3.,  11.,  33.,   1.,   7., 136.,   5.,   5.,
0.,   0.,   6.],
       [  3.,  68.,  52.,   1.,  24., 116.,   1.,  29.,
0.,   1.,   2.],
       [  2.,  33.,  33.,   0.,  12.,  33.,   2.,   0.,
0.,   1.,   1.],
       [  2.,  23.,  30.,   1.,   9.,  30.,   1.,   2.,
0.,   0.,   4.]])
```

In [180]:

```
y = data['custcat'].values
y[0:5]
```

Out[180]:

```
array([1, 4, 3, 1, 3])
```

In [181]:

```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

Out[181]:

```
array([[-0.03, -1.06,  0.18,  1.01, -0.25, -0.13,  1.09, -
0.59, -0.22,
        -1.03, -0.23],
       [ 1.2 , -1.15, -0.69,  1.01, -0.45,  0.55,  1.91, -
0.59, -0.22,
        -1.03,  2.56],
       [ 1.2 ,  1.52,  0.82,  1.01,  1.23,  0.36, -1.37,
1.79, -0.22,
         0.97, -0.23],
       [-0.03, -0.12, -0.69, -0.99,  0.04, -0.42, -0.55, -
1.09, -0.22,
         0.97, -0.93],
       [-0.03, -0.59, -0.93,  1.01, -0.25, -0.44, -1.37, -
0.89, -0.22,
        -1.03,  1.16]])
```

2. Split Data to Train and Test sets

In [182]:

```
# split X and y into training and testing sets will help to have more c
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random
```

Classification -K Nearest Neighbors(KNN)

In [183]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classific
k = 4
#Train Model and Predict
neigh4 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
y_pred = neigh4.predict(X_test)
```

In [184]:

```
# Preciision, recall, f-score from the multi-class support function
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.31      0.45      0.37        51
           2       0.33      0.36      0.35        44
           3       0.33      0.30      0.31        54
           4       0.31      0.18      0.23        51

    accuracy                           0.32       200
   macro avg       0.32      0.32      0.31       200
weighted avg       0.32      0.32      0.31       200
```

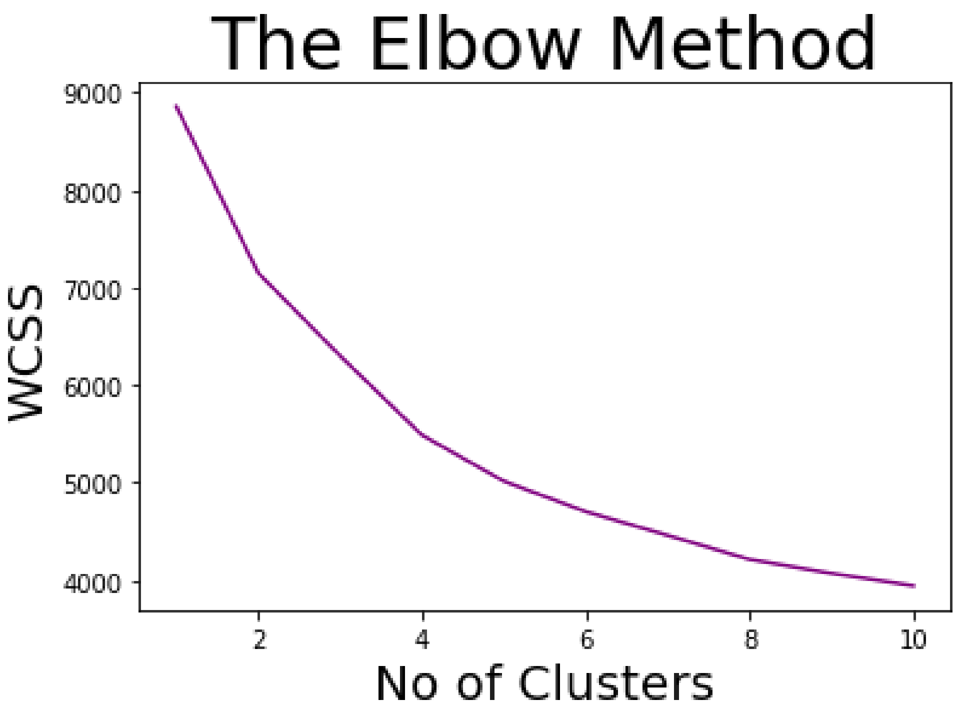
In [185]:

```
k = 6
neigh6 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat6 = neigh6.predict(X_test)

# Preciision, recall, f-score from the multi-class support function
print(classification_report(y_test, yhat6))
```

```
              precision    recall  f1-score   support

           1       0.33      0.47      0.39        51
           2       0.30      0.32      0.31        44
           3       0.30      0.28      0.29        54
           4       0.30      0.18      0.22        51

    accuracy                           0.31       200
   macro avg       0.31      0.31      0.30       200
weighted avg       0.31      0.31      0.30       200
```

In [186]:

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    km = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_i
    km.fit(X_train)
    wcss.append(km.inertia_)

plt.plot(range(1, 11), wcss, c="purple")
plt.title('The Elbow Method', fontsize = 30)
plt.xlabel('No of Clusters', fontsize = 20)
plt.ylabel('WCSS', fontsize = 20)
plt.show()
```



In [187]:

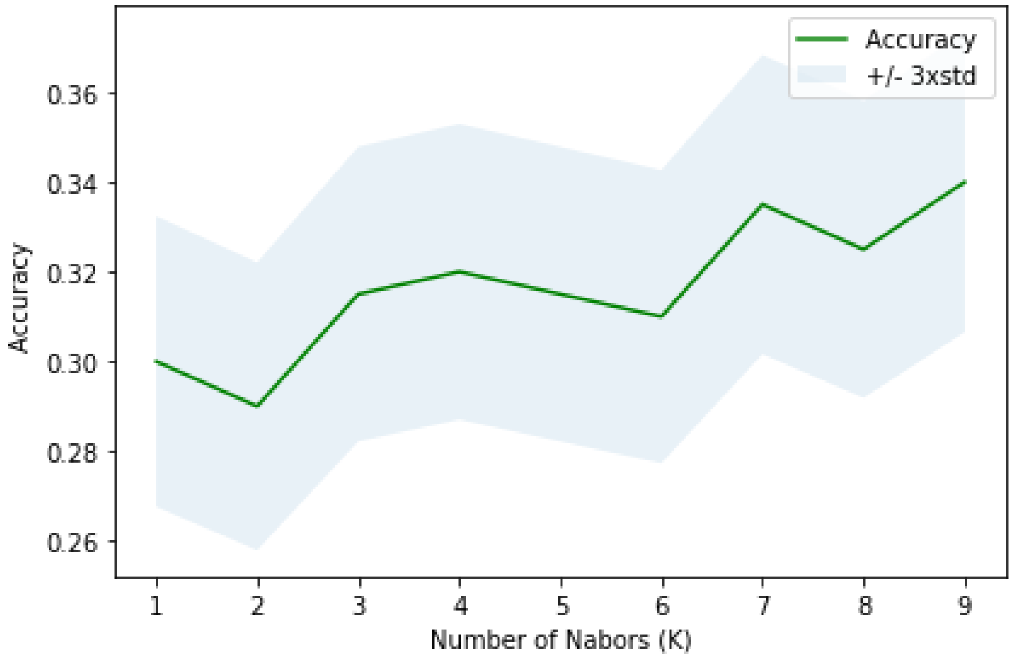
```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
f1_scores = list()
error_rates = list()
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)
    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

Out[187]:

```
array([0.3 , 0.29, 0.32, 0.32, 0.32, 0.31, 0.34, 0.33, 0.3
4])
```

In [188]:

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```



In [189]:

```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_ac
```

```
The best accuracy was with 0.34 with k= 9
```

Summary :

```
We got best accuracy with K =9.
```

In [ ]: