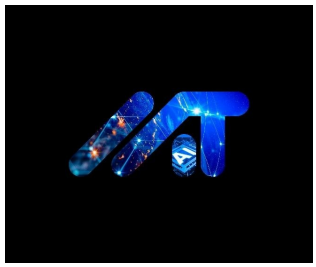
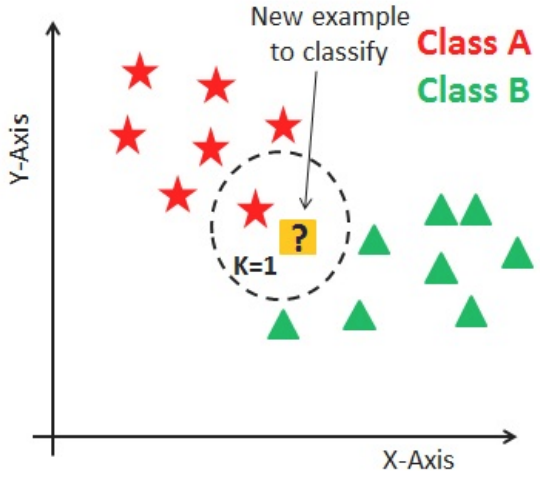


K-Nearest Neighbors

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

Here's an visualization of the K-Nearest Neighbors algorithm.



In this case, we have data points of Class A and B. We want to predict what the question mark box (test data point) is. If we consider a k value of 1 (1 nearest data point), we will obtain a prediction of Class A.

In this sense, it is important to consider the value of k. Hopefully from this diagram, you should get a sense of what the K-Nearest Neighbors algorithm is. It considers the 'K' Nearest Neighbors (data points) when it predicts the classification of the test point.

Importing required packages

In [26]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.model_selection import train_test_split
%matplotlib inline
```

Let's download and import the data on China's GDP using *pandas* `read_csv()` method.

Download Dataset

Understanding the Data

`telecomData.csv`:

Let's imagine telecommunications provider has segmented its customer base by service usage patterns, categorizing the customers into four groups. If demographic data can be used to predict group membership, the company can customize offers for individual prospective customers. It is a classification problem. That is, given the dataset, with predefined labels, we need to build a model to 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 1st Basic Service, 2nd E-Service, 3rd Plus Service and 4thTotal 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

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.

Reading the data

In [3]:
```
df = pd.read_csv("telecomData.csv")

# take a look at the dataset
df.head()
```

Out[3]:

	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	2	13	44	1	9	64.0	4	5	0.0	0	2	1
1	3	11	33	1	7	136.0	5	5	0.0	0	6	4
2	3	68	52	1	24	116.0	1	29	0.0	1	2	3
3	2	33	33	0	12	33.0	2	0	0.0	1	1	1
4	2	23	30	1	9	30.0	1	2	0.0	0	4	3

Data Exploration

Let's first have a descriptive exploration on our data.

In [4]:
```
df.describe()
```

Out[4]:

	region	tenure	age	marital	address	income	ed	employ	retire	gender	
count	1000.0000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	100
mean	2.0220	35.526000	41.684000	0.495000	11.551000	77.535000	2.671000	10.987000	0.047000	0.517000	
std	0.8162	21.359812	12.558816	0.500225	10.086681	107.044165	1.222397	10.082087	0.211745	0.499961	
min	1.0000	1.000000	18.000000	0.000000	0.000000	9.000000	1.000000	0.000000	0.000000	0.000000	
25%	1.0000	17.000000	32.000000	0.000000	3.000000	29.000000	2.000000	3.000000	0.000000	0.000000	
50%	2.0000	34.000000	40.000000	0.000000	9.000000	47.000000	3.000000	8.000000	0.000000	1.000000	
75%	3.0000	54.000000	51.000000	1.000000	18.000000	83.000000	4.000000	17.000000	0.000000	1.000000	
max	3.0000	72.000000	77.000000	1.000000	55.000000	1668.000000	5.000000	47.000000	1.000000	1.000000	

Data Visualization and Analysis

Let's see how many of each class is in our data set

In [5]:
```
df['custcat'].value_counts()
```

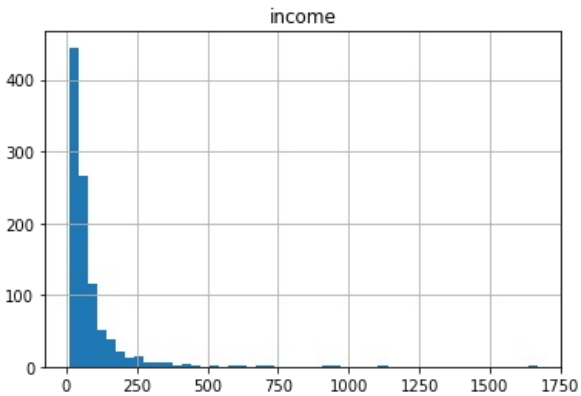
Out[5]:
```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

281 Plus Service, 266 Basic-service, 236 Total Service, and 217 E-Service customers

You can easily explore your data using visualization techniques:

In [6]:
```
df.hist(column='income', bins=50)
```

Out[6]:
```
array([[<AxesSubplot:title={'center':'income'}>]], dtype=object)
```

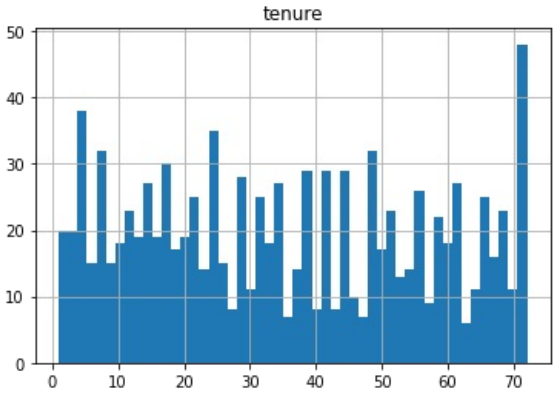


In [7]:
```
df.columns
```

Out[7]:
```
Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside', 'custcat'],
      dtype='object')
```

In [8]:
```
df.hist(column='tenure', bins=50)
```

Out[8]:
```
array([[<AxesSubplot:title={'center':'tenure'}>]], dtype=object)
```



In [11]:
```
X = df[['region', 'tenure','age', 'marital', 'address', 'income', 'ed', 'employ','retire', 'gender', 'reside']]
```

In [10]:
```
y = df['custcat'].values
```

Normalization

Data Standardization gives the data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on the distance of data points

In [15]:
```
X = StandardScaler().fit(X).transform(X.astype(float))
```

In [16]:
```
type(X)
```

Out[16]:
```
numpy.ndarray
```

Creating train and test dataset

Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the model. Therefore, it gives us a better understanding of how well our model generalizes on new data.

We know the outcome of each data point in the testing dataset, making it great to test with! Since this data has not been used to train the model, the model has no knowledge of the outcome of these data points. So, in essence, it is truly an out-of-sample testing.

Let's split our dataset into train and test sets. Around 80% of the entire dataset will be used for training and 20% for testing.

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (800, 11) (800,)
Test set: (200, 11) (200,)
```

Classification

In [19]:
```
k = 4
#Train Model and Predict
model = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
model
```

Out[19]:
```
KNeighborsClassifier(n_neighbors=4)
```

In [20]:
```
y_pred = model.predict(X_test)
```

Evaluation

In [36]:
```
print("Train set Accuracy: ", accuracy_score(y_train, model.predict(X_train)))
print("Test set Accuracy: ", accuracy_score(y_test, y_pred))
```

```
Train set Accuracy:  0.45125
Test set Accuracy:  0.345
```

Exercise

Try to fit KNN with k=8 with the dataset

▶ Click here for the solution

What about other K?

K in KNN, is the number of nearest neighbors to examine. It is supposed to be specified by the user. So, how can we choose right value for K? The general solution is to reserve a part of your data for testing the accuracy of the model. Then choose k =1, use the training part for modeling, and calculate the accuracy of prediction using all samples in your test set. Repeat this process, increasing the k, and see which k is the best for your model.

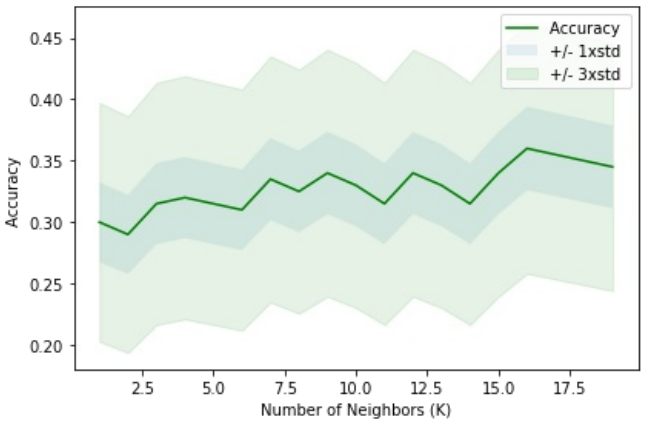
We can calculate the accuracy of KNN for different values of k.

In [33]:
```
Ks = 20
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))

for n in range(1,Ks):

    #Train Model and Predict
    model = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    y_pred=model.predict(X_test)
    mean_acc[n-1] = accuracy_score(y_test, y_pred)


    std_acc[n-1]=np.std(y_pred==y_test)/np.sqrt(y_pred.shape[0])

mean_acc
```

Out[33]:
```
array([0.3  , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ,
       0.33 , 0.315, 0.34 , 0.33 , 0.315, 0.34 , 0.36 , 0.355, 0.35 ,
       0.345])
```

Plot the model accuracy for a different number of neighbors.

In [34]:
```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.fill_between(range(1,Ks),mean_acc - 3 * std_acc,mean_acc + 3 * std_acc, alpha=0.10,color="green")
plt.legend(('Accuracy ', '+/- 1xstd','+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [35]:
```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

```
The best accuracy was with 0.36 with k= 16
```

Thank you

Author

Moazzam Ali

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js