In [1]:
```
import wget as wget
```

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import numpy as no
from sklearn import preprocessing
%matplotlib inline
```

Downloading Data

We will use wget to download the dataset. Below is the example of using subprocess and runcmd. The process for wget will look similar.

In [4]:
```
import subprocess

def runcmd(cmd, verbose = False, *args, **kwargs):

    process = subprocess.Popen(
        cmd,
        stdout = subprocess.PIPE,
        stderr = subprocess.PIPE,
        text = True,
        shell = True
    )
    std_out, std_err = process.communicate()
    if verbose:
        print(std_out.strip(), std_err)
    pass

runcmd('echo "Hello, World!"', verbose = True)
```

```
"Hello, World!"
```

In [5]:
```
runcmd("wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
```

```
 --2022-10-17 17:58:42--  https://cf-courses-data.s3.us.cloud-object-storage.appdom
ain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/tel
eCust1000t.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-da
ta.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-course
s-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:443... connecte
d.
HTTP request sent, awaiting response... 200 OK
Length: 36047 (35K) [text/csv]
Saving to: 'teleCust1000t.csv.4'

     0K .......... .......... .......... .....                100%  117K=0,3s

2022-10-17 17:58:43 (117 KB/s) - 'teleCust1000t.csv.4' saved [36047/36047]
```

Load Data

The data is from a telecommunications company which provides data by segmenting its customers by service usage. There are 4 groups and it is shown below. We would like to know whether individual demographic features can be used to identify the types of packages that would be offered to prospective customers. It is also possible that new classification/s of unknown emerged. The features of individuals can be seen below.

In [6]:
```
df=pd.read_csv('teleCust1000t.csv')
df.head()
```

Out[6]:

	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

See how many of each class is in our data set.

In [7]:
```
df['custcat'].value_counts()
```

Out[7]:
```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

Plus Service: 281

Basic Service: 266

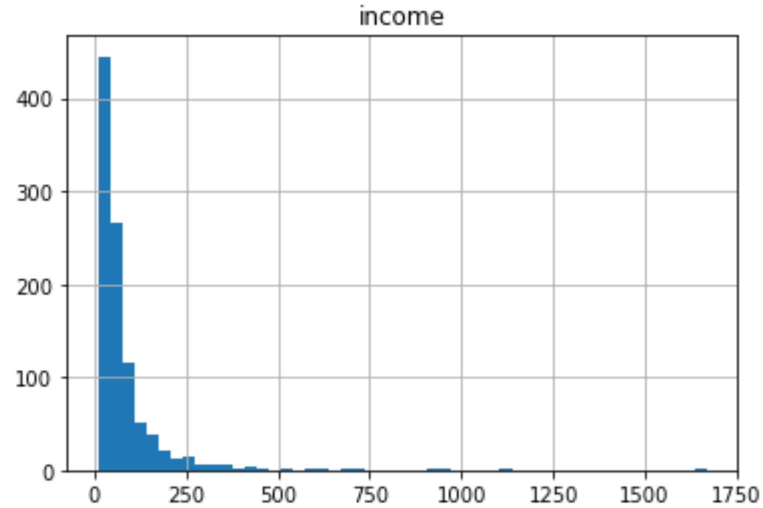
Total Service: 236

E-Serice: 2017

Let's try visualizing

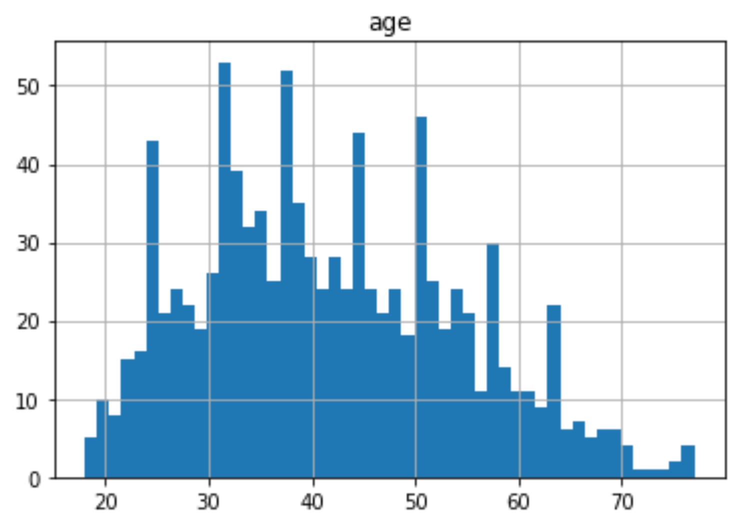
In [8]:
```
df.hist(column='income', bins=50)
```

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



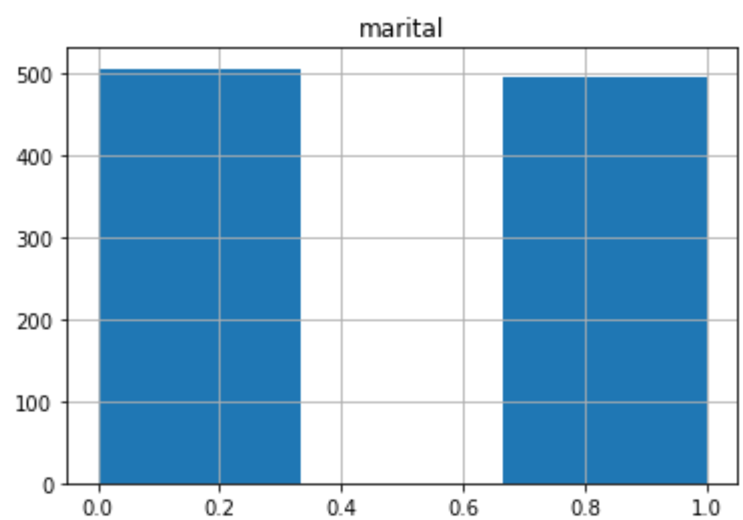
In [9]:
```
df.hist(column='age', bins=50)
```

Out[9]:
```
array([[<AxesSubplot:title={'center':'age'}>]], dtype=object)
```



In [10]:
```
df.hist(column='marital', bins=3)
```

Out[10]:
```
array([[<AxesSubplot:title={'center':'marital'}>]], dtype=object)
```



Feature Set

Identify the feature sets as X:

In [11]:
```
df.columns
```

Out[11]:
```
Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside', 'custcat'],
      dtype='object')
```

Converting Pandas to a Numpy array to use scikit-learn:

In [12]:
```
X=df[['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside']] .values
X[0:5]
```

Out[12]:
```
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 [13]:
```
y=df['custcat'].values
y[0:5]
```

Out[13]:
```
array([1, 4, 3, 1, 3], dtype=int64)
```

Normalizing the Data

Normalizing the data gives zero mean and unit variance. KKN algorithm based itself on the distance of data points, making normalization as a good practice:

In [14]:
```
X=preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

Out[14]:
```
array([[-0.02696767, -1.055125  ,  0.18450456,  1.0100505 , -0.25303431,
        -0.12650641,  1.0877526 , -0.5941226 , -0.22207644, -1.03459817,
        -0.23065004],
       [ 1.19883553, -1.14880563, -0.69181243,  1.0100505 , -0.4514148 ,
         0.54644972,  1.9062271 , -0.5941226 , -0.22207644, -1.03459817,
         2.55666158],
       [ 1.19883553,  1.52109247,  0.82182601,  1.0100505 ,  1.23481934,
         0.35951747, -1.36767088,  1.78752803, -0.22207644,  0.96655883,
        -0.23065004],
       [-0.02696767, -0.11831864, -0.69181243, -0.9900495 ,  0.04453642,
        -0.41625141, -0.54919639, -1.09029981, -0.22207644,  0.96655883,
        -0.92747794],
       [-0.02696767, -0.58672182, -0.93080797,  1.0100505 , -0.25303431,
        -0.44429125, -1.36767088, -0.89182893, -0.22207644, -1.03459817,
         1.16300577]])
```

Train Test Split

Separate the data into train and test. This is to avoid overfitting and the whether the data works good for out of sample estimation.

In [15]:
```
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_st
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

K nearest Neighbour (KNN)

Importing library

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [17]:
```
k=4
#train and predict
neigh=KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
neigh
```

Out[17]:
```
▾       KNeighborsClassifier

KNeighborsClassifier(n_neighbors=4)
```

Predicting

Use model to predict test set

In [18]:
```
yhat=neigh.predict(X_test)
yhat[0:5]
```

Out[18]:
```
array([1, 1, 3, 2, 4], dtype=int64)
```

Evaluating the accuracy

In [19]:
```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.5475
Test set Accuracy:  0.32
```

Let's try using other k

In [20]:
```
k=6
#train and predict
neigh=KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
neigh
```

Out[20]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=6)
```

In [21]:
```
yhat=neigh.predict(X_test)
yhat[0:5]
```

Out[21]:
```
array([3, 3, 3, 4, 4], dtype=int64)
```

In [22]:
```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.51625
Test set Accuracy:  0.31
```

Test set accuracy is getting better, but the Train set does not.

Question to note: How can we choose the right number of Ks?

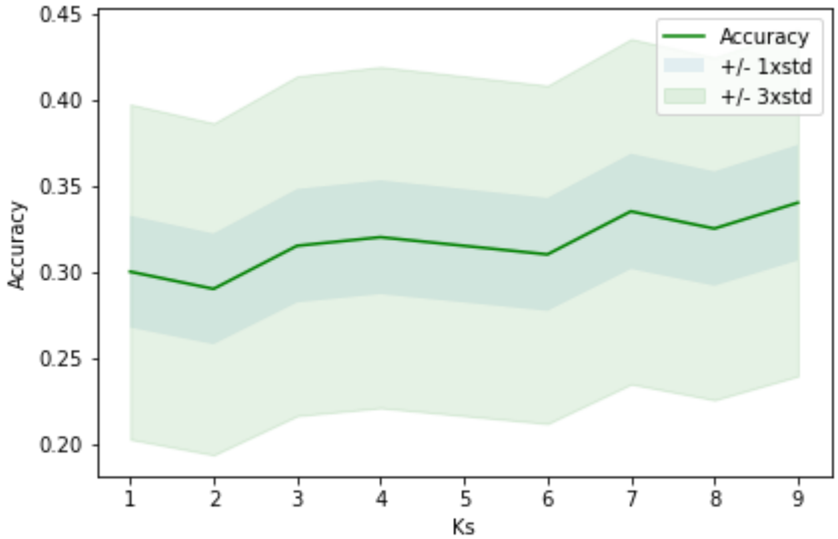
The K should be specified by the user, but the general solution is to set aside parts of the data for testing the accuracy of the model. Do the same process above starting from k=1 and increase it. See which k presents the best accuracy.

We can also calculate the accuracy of KNN for different ks.

In [23]:
```
Ks=10

mean_acc=np.zeros((Ks-1))
std_acc=np.zeros((Ks-1))

for n in range(1,Ks):

    #Train the 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[23]:
```
array([0.3  , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ])
```

In [24]:
```
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=0.1)
plt.fill_between(range(1,Ks), mean_acc-3*std_acc, mean_acc+3*std_acc, alpha=0.1, co
plt.legend(('Accuracy', '+/- 1xstd', '+/- 3xstd'))
plt.ylabel('Accuracy')
plt.xlabel('Ks')
plt.tight_layout()
plt.show()
```



In [25]:
```
print("The best accuracy for K:",mean_acc.argmax()+1, "with accuracy:", mean_acc.ma
```

```
The best accuracy for K: 9 with accuracy: 0.34
```