k means culstring with titanic dataset

In [8]:
```
# ================================================================================
# Setting the Envoirnment
# ================================================================================

import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from scipy.spatial.distance import cdist
import seaborn as sns
import matplotlib.pyplot as plt
#%matplotlib inline
import os

os.chdir(r'C:\Users\dell\Desktop')
train = pd.read_csv("train.csv")
```

In [9]:
```
# ================================================================================
# Exploratory Data Analysis
# ================================================================================
print(train.head())
train_stat = pd.DataFrame(train.describe()).reset_index()
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

In [10]:
```
#Checking missing values
print(train.isna().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
```

```
Cabin          687
Embarked         2
dtype: int64
```

In [11]:
```
#Imputing the Missing Values
train.fillna(train.mean(), inplace=True)
```

In [13]:
```
#Survival count with respect to Pclass:
Surv_Pclass=train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().so
print(Surv_Pclass)
```

```
   Pclass  Survived
0       1  0.629630
1       2  0.472826
2       3  0.242363
```

In [15]:
```
#Survival count with respect to Gender:
Surv_Gen=train[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values
print(Surv_Gen)
```

```
      Sex  Survived
0  female  0.742038
1    male  0.188908
```

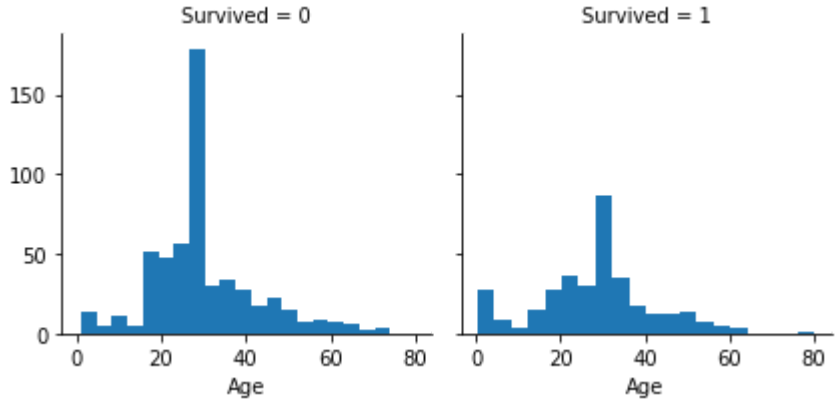
In [16]:
```
#Survival count with respect to SibSp:
Surv_SibSp = train[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sor
print(Surv_SibSp)
```

```
   SibSp  Survived
1      1  0.535885
2      2  0.464286
0      0  0.345395
3      3  0.250000
4      4  0.166667
5      5  0.000000
6      8  0.000000
```

Data Visualizations

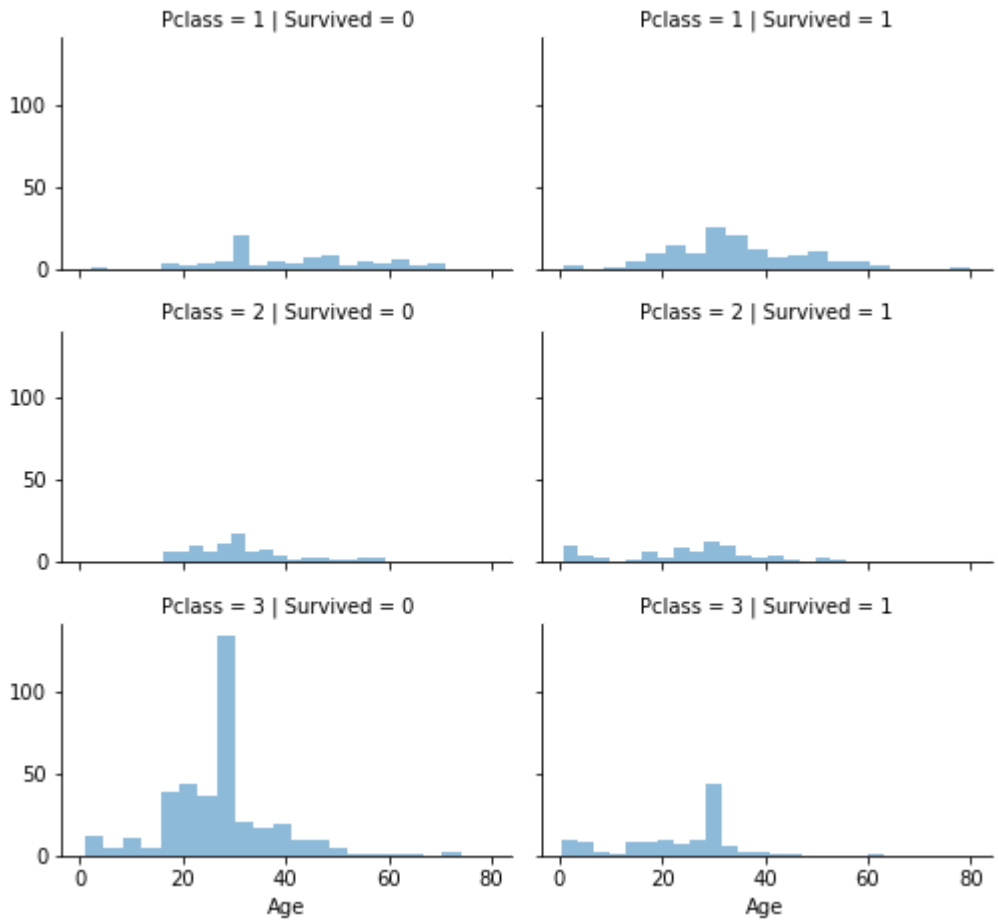
In [17]:
```
#Histogram of survived wrt to age
g = sns.FacetGrid(train, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

Out[17]:
```
<seaborn.axisgrid.FacetGrid at 0x53fa880>
```



In [19]:
```
#Histogram of survived wrt to plcass
grid = sns.FacetGrid(train, col='Survived', row='Pclass', height=2.2, aspect=1.6)
```

```
grid.map(plt.hist, 'Age', alpha=0.5, bins=20)
grid.add_legend();
```



Converting Categorical Features into Numeric

In [25]:
```
labelEncoder = LabelEncoder()
labelEncoder.fit(train['Sex'])
train['Sex'] = labelEncoder.transform(train['Sex'])
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Sex          891 non-null    int32
 4   Age          891 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
dtypes: float64(2), int32(1), int64(5)
memory usage: 52.3 KB
```

Building the K-Means Model

```
#Dropping the Survival Feature
```

In [30]:
```
X = np.array(train.drop(['Survived'], 1).astype(float))

y = np.array(train['Survived'])

kmeans = KMeans(n_clusters=2) # You want cluster the passenger records into 2: Survived
kmeans.fit(X)


kmeans=KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=900,
    n_clusters=2, n_init=10,
    random_state=0, tol=0.0001, verbose=0)

kmeans.fit(X)


#algorithm = auto, elkan, full
#k-means++ ensures the Model is converged,
```

Out[30]:
```
KMeans(max_iter=900, n_clusters=2, random_state=0)
```

Evaluating the Clusters & Scaling the Data

In [37]:
```
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

kmeans.fit(X_scaled)

KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=900,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=0, tol=0.0001, verbose=0)

kmeans.fit(X_scaled)

correct = 0
for i in range(len(X_scaled)):
    predict_me = np.array(X_scaled[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print("Accuracy of Kmeans is " + str(correct/len(X_scaled)))

kmeans = KMeans(n_clusters=2) # You want cluster the passenger records into 2: Survived
#kmeans.fit(X)
```

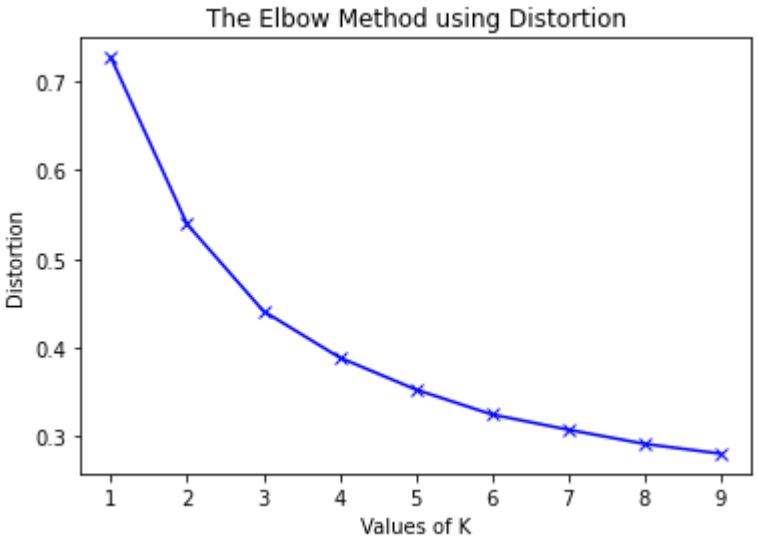
```
Accuracy of Kmeans is 0.7867564534231201
```

Finding the Optimal Clusters through Elbow Method

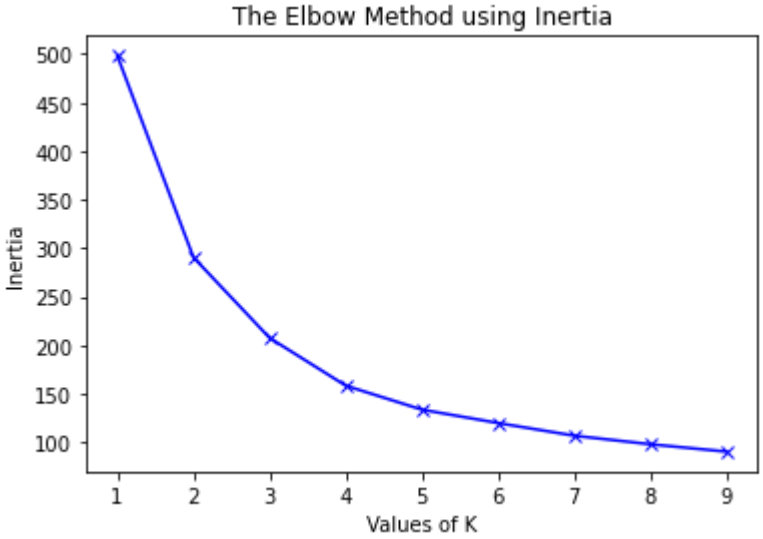
In [39]:
```
distortions = [] #It is calculated as the average of the squared distances from the clu
inertias = [] # It is the sum of squared distances of samples to their closest cluster
mapping1 = {}
mapping2 = {}
K = range(1,10)
```

```
#Based on Distortion
for k in K:
    #Building and fitting the model
    #i=1
    print("performing the clustering for k"+ str(k))
    kmeanModel = KMeans(n_clusters=k).fit(X_scaled)
    kmeanModel.fit(X_scaled)

    distortions.append(sum(np.min(cdist(X_scaled, kmeanModel.cluster_centers_,
                      'euclidean'),axis=1)) / X_scaled.shape[0])
    inertias.append(kmeanModel.inertia_)

    mapping1[k] = sum(np.min(cdist(X_scaled, kmeanModel.cluster_centers_,
                 'euclidean'),axis=1)) / X_scaled.shape[0]
    mapping2[k] = kmeanModel.inertia_


for key,val in mapping1.items():
    print(str(key)+' : '+str(val))

plt.plot(K, distortions, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Distortion')
plt.title('The Elbow Method using Distortion')
plt.show()

#Based on Inertia
for key,val in mapping2.items():
    print(str(key)+' : '+str(val))

plt.plot(K, inertias, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```

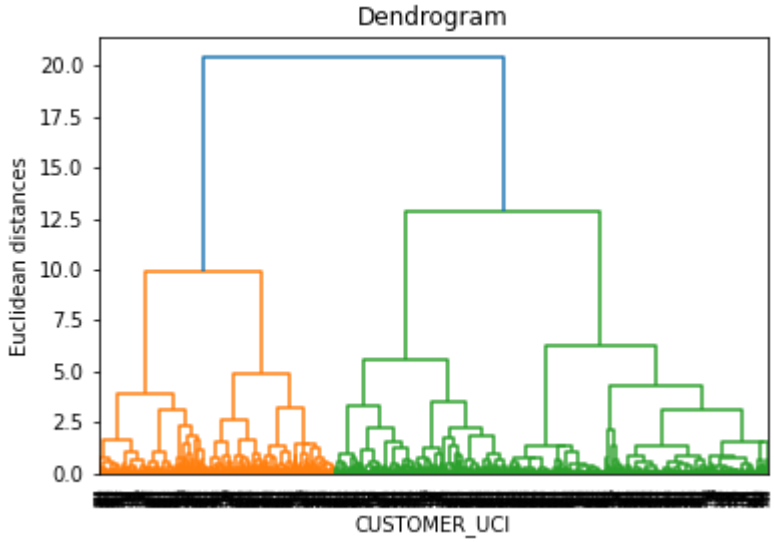
```
performing the clustering for k1
performing the clustering for k2
performing the clustering for k3
performing the clustering for k4
performing the clustering for k5
performing the clustering for k6
performing the clustering for k7
performing the clustering for k8
performing the clustering for k9
1 : 0.7261849386188889
2 : 0.5388274622064898
3 : 0.4413432381092249
4 : 0.3891020987311309
5 : 0.35310600765441197
6 : 0.32507292790990255
7 : 0.30797634805701674
8 : 0.2920523768740757
9 : 0.28102227155839704
```




```
1 : 498.3941712954537
2 : 290.57192641064
3 : 207.6105155112092
4 : 158.40740289134195
5 : 133.70779757192315
6 : 119.94476912548994
7 : 106.88949762507814
8 : 98.09223352088136
9 : 90.48389976271896
```



In [40]:
```
# Heirarchical Clustering
import scipy.cluster.hierarchy as sch

dendrogram = sch.dendrogram(sch.linkage(X_scaled, method  = "ward"))
plt.title('Dendrogram')
plt.xlabel('CUSTOMER_UCI')
plt.ylabel('Euclidean distances')
plt.show()
```


Dendrogram
Euclidean distances
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
CUSTOMER_UCI

In [ ]: