PCAdatatoyMoviesClustering

September 14, 2019

In [2]:
```
#Extracción de Datos
import pandas as pd
import numpy as np
dataset = "datatoy.csv"
df = pd.read_csv(dataset)
df.head()
df
```

Out[2]:

	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10	i11	i12	i13	i14	i15
0	5	5	5	5	5	0	1	0	0	0	1	0	0	0	0
1	5	5	5	5	5	0	0	1	0	0	0	0	0	0	0
2	5	5	0	5	5	0	0	0	0	0	0	3	0	0	0
3	0	0	1	0	0	5	5	5	5	5	0	1	0	2	0
4	0	3	0	0	2	5	5	0	5	5	0	2	0	4	0
5	0	1	0	0	0	5	5	5	5	5	0	0	0	0	0
6	0	0	0	4	0	0	0	0	1	0	5	4	5	5	5
7	0	1	0	0	0	0	4	0	0	0	5	5	5	4	5
8	0	0	0	0	0	3	0	0	3	0	5	4	5	5	5
9	5	5	5	5	5	1	5	2	1	5	0	0	0	0	0
10	0	1	0	0	0	5	5	5	5	5	0	0	5	0	0

In [3]:
```
#Descripción de datos
df2=df.describe()
df2
```

Out[3]:

	i1	i2	i3	i4	i5	i6	\
count	11.000000	11.000000	11.000000	11.000000	11.00000	11.000000	
mean	1.818182	2.363636	1.454545	2.181818	2.00000	2.181818	
std	2.522625	2.248232	2.296242	2.522625	2.44949	2.400757	
min	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	
25%	0.000000	0.500000	0.000000	0.000000	0.00000	0.000000	
50%	0.000000	1.000000	0.000000	0.000000	0.00000	1.000000	
75%	5.000000	5.000000	3.000000	5.000000	5.00000	5.000000	
max	5.000000	5.000000	5.000000	5.000000	5.00000	5.000000	

	i7	i8	i9	i10	i11	i12	\
count	11.000000	11.000000	11.000000	11.000000	11.000000	11.000000	
mean	2.727273	1.636364	2.272727	2.272727	1.454545	1.727273	

```
std     2.453198   2.248232   2.327699   2.611165   2.296242   1.954017
min     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
25%     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
50%     4.000000   0.000000   1.000000   0.000000   0.000000   1.000000
75%     5.000000   3.500000   5.000000   5.000000   3.000000   3.500000
max     5.000000   5.000000   5.000000   5.000000   5.000000   5.000000

              i13        i14        i15
count   11.000000  11.000000  11.000000
mean     1.818182   1.818182   1.363636
std      2.522625   2.227922   2.335497
min      0.000000   0.000000   0.000000
25%      0.000000   0.000000   0.000000
50%      0.000000   0.000000   0.000000
75%      5.000000   4.000000   2.500000
max      5.000000   5.000000   5.000000
```

In [4]:
```
# Estandarización de datos en una escala (mean = 0 and variance = 1) que es requerida p
# muchos algoritmos de machine learning.
import math
from sklearn.preprocessing import StandardScaler
x=df
x = StandardScaler().fit_transform(x)
print(x)
print("Se han estandarizado los datos")
```

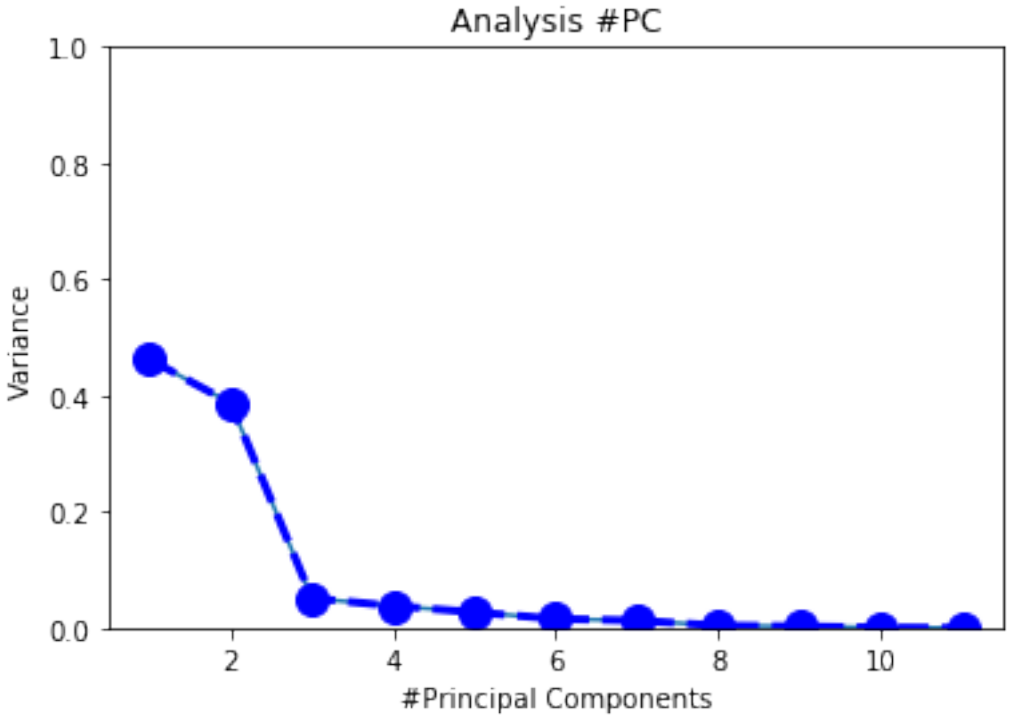
```
[[ 1.32287566  1.22987395  1.61938686  1.17168987  1.28452326 -0.95316176
  -0.73845597 -0.76337004 -1.0240399  -0.91287093 -0.2076137  -0.92710507
  -0.75592895 -0.85592099 -0.61237244]
 [ 1.32287566  1.22987395  1.61938686  1.17168987  1.28452326 -0.95316176
  -1.16598311 -0.29686613 -1.0240399  -0.91287093 -0.66436384 -0.92710507
  -0.75592895 -0.85592099 -0.61237244]
 [ 1.32287566  1.22987395 -0.66436384  1.17168987  1.28452326 -0.95316176
  -1.16598311 -0.76337004 -1.0240399  -0.91287093 -0.66436384  0.68313005
  -0.75592895 -0.85592099 -0.61237244]
 [-0.75592895 -1.10264561 -0.2076137  -0.90711474 -0.85634884  1.23116728
   0.97165259  1.56914952  1.22884788  1.09544512 -0.66436384 -0.39036003
  -0.75592895  0.0855921  -0.61237244]
 [-0.75592895  0.29686613 -0.66436384 -0.90711474  0.          1.23116728
   0.97165259 -0.76337004  1.22884788  1.09544512 -0.66436384  0.14638501
  -0.75592895  1.02710518 -0.61237244]
 [-0.75592895 -0.6361417  -0.66436384 -0.90711474 -0.85634884  1.23116728
   0.97165259  1.56914952  1.22884788  1.09544512 -0.66436384 -0.92710507
  -0.75592895 -0.85592099 -0.61237244]
 [-0.75592895 -1.10264561 -0.66436384  0.75592895 -0.85634884 -0.95316176
  -1.16598311 -0.76337004 -0.57346234 -0.91287093  1.61938686  1.21987509
   1.32287566  1.49786172  1.63299316]
 [-0.75592895 -0.6361417  -0.66436384 -0.90711474 -0.85634884 -0.95316176
```

```
   0.54412545 -0.76337004 -1.0240399  -0.91287093  1.61938686  1.75662013
   1.32287566  1.02710518  1.63299316]
 [-0.75592895 -1.10264561 -0.66436384 -0.90711474 -0.85634884  0.35743566
  -1.16598311 -0.76337004  0.32769277 -0.91287093  1.61938686  1.21987509
   1.32287566  1.49786172  1.63299316]
 [ 1.32287566  1.22987395  1.61938686  1.17168987  1.28452326 -0.51629596
   0.97165259  0.16963779 -0.57346234  1.09544512 -0.66436384 -0.92710507
  -0.75592895 -0.85592099 -0.61237244]
 [-0.75592895 -0.6361417  -0.66436384 -0.90711474 -0.85634884  1.23116728
   0.97165259  1.56914952  1.22884788  1.09544512 -0.66436384 -0.92710507
   1.32287566 -0.85592099 -0.61237244]]
Se han estandarizado los datos


C:\Users\UPS\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:645: DataConversionWarni
  return self.partial_fit(X, y)
C:\Users\UPS\Anaconda3\lib\site-packages\sklearn\base.py:464: DataConversionWarning: Data with
  return self.fit(X, **fit_params).transform(X)
```

In [7]:
```
#Reducción de Dimensionalidad con PCA: un dataset con menores dimensiones necesita men
from sklearn.decomposition import PCA

#Si no se especifica el número de componentes en PCA, se intenta con todas las caracte
pca = PCA()
principalComponents = pca.fit_transform(x)
principalComponents

#La varianza explicada dice cuanta información (varianza) puede ser obtenida a cada co
#Esto es importante ya que si bien se puede convertir el espacio de 4 dimensiones en e
#parte de la varianza (información). Al usar explained_variance_ratio_ se puede ver qu
#contiene 47% de la varianza y el segundo componente principal contiene 38% de la vari
#contienen el 85% de la información.
num_components=principalComponents.shape[1]
num_components
explained_variance_ratio_=pca.explained_variance_ratio_
explained_variance_ratio_
print('Varianzas:')
print(np.around(explained_variance_ratio_, decimals=3))

a = range(1,num_components+1)
num_pc= a[::1]

#Visualizar en 2D
import matplotlib.pyplot as plt

plt.plot(num_pc, explained_variance_ratio_)
plt.plot(num_pc, explained_variance_ratio_, color='blue', linestyle='dashed', linewidth
```

```
                marker='o', markerfacecolor='blue', markersize=12)
        plt.ylim((0,1))
        plt.title('Analysis #PC')
        plt.xlabel('#Principal Components')
        plt.ylabel('Variance')
        plt.show()
```

```
Varianzas:
[0.462 0.387 0.051 0.038 0.027 0.016 0.013 0.004 0.003 0.001 0.   ]
```

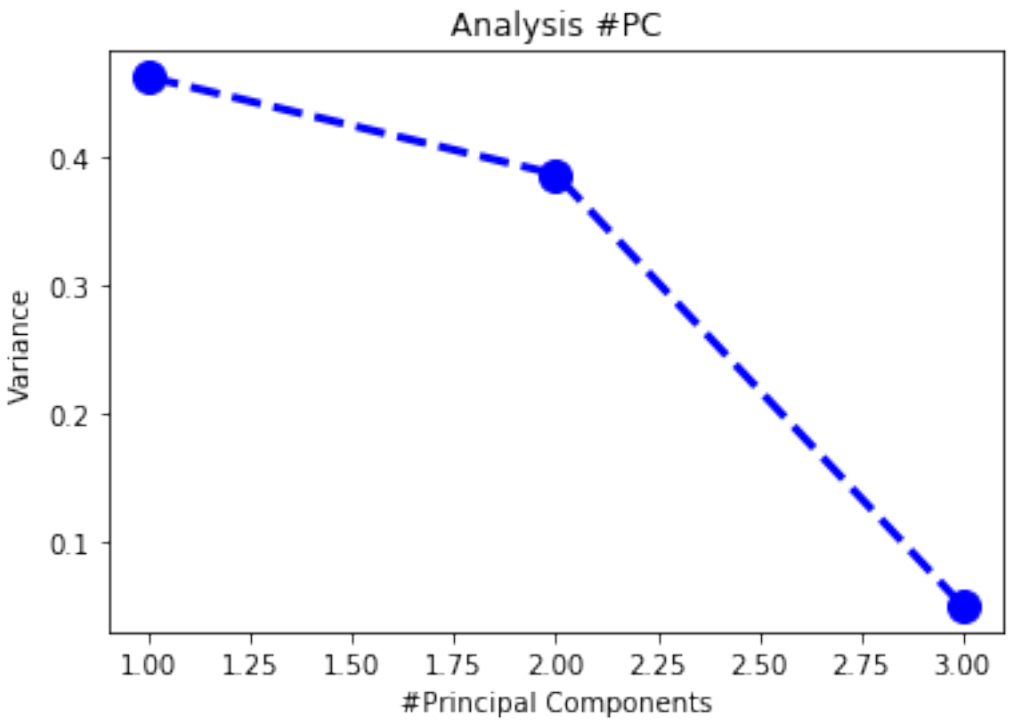


In [9]:
```
        #A partir de los resultados anteriores y de la gráfica anterior se puede deducir que s
        #Con más componentes no se obtiene un incremento relevante en la cantidad de informaci
        #Utilizar 3 componentes o 2 componentes posibilita incluso la visualización de los com
        num_components=3
        pca = PCA(num_components)
        principalComponents = pca.fit_transform(x)

        explained_variance_ratio_=pca.explained_variance_ratio_
        explained_variance_ratio_
        print('Varianzas:')
        print(np.around(explained_variance_ratio_, decimals=3))
        a = range(num_components)
        num_pc= a[::1]
```

```
        principalDf = pd.DataFrame(data = principalComponents
                     , columns = num_pc)

        principalDf=round(principalDf, 2)
        print(principalDf)

        #medidas=principalDf.describe()
        #medidas=round(medidas,5)
        #medidas
```

```
Varianzas:
[0.462 0.387 0.051]
       0     1     2
0   3.73  0.90 -0.30
1   3.83  0.72 -0.49
2   2.83  1.40  0.92
3  -1.67 -2.96 -0.06
4  -1.16 -1.73  2.29
5  -1.28 -3.32 -0.33
6  -2.08  3.54 -0.49
7  -2.52  2.95 -0.12
8  -3.09  2.58  0.08
9   3.22 -1.17 -0.26
10 -1.82 -2.91 -1.24
```

```
In [10]: #Explicación de la varianza
         explained_variance_ratio_
         a = range(num_components)
         num_pc= np.linspace(1,num_components,num_components)

         #Visualización en 2D
         import matplotlib.pyplot as plt

         #plt.plot(num_pc, explained_variance)
         plt.plot(num_pc, explained_variance_ratio_, color='blue', linestyle='dashed', linewid
                  marker='o', markerfacecolor='blue', markersize=12)

         #plt.ylim((0,0.4))
         plt.title('Analysis #PC')
         plt.xlabel('#Principal Components')
         plt.ylabel('Variance')
         plt.show()
```



In [11]:
```
#Análisis de la correlación entre los componentes principales con la función: corr
#En una técnica de reducción de dimensionalidad se busca la información más relevante
#correlación aproximadamente igual a cero, puesto que el objetivo es que entre los co
#puesto que si dos componentes fueran similares, uno de ellos sería redundante.

#DataFrame.corr(method='pearson', min_periods=1)
#Compute pairwise correlation of columns, excluding NA/null values.

#method : {pearson, kendall, spearman} or callable
#pearson : standard correlation coefficient
#kendall : Kendall Tau correlation coefficient
#spearman : Spearman rank correlation

#Correlación entre componentes
correl=principalDf.corr()
correl=round(correl,5)
correl

#Correlación entre usuarios
#principalDf=np.transpose(principalDf)
#principalComponents.reverse()
#print(principalDf)
#correl=principalDf.corr(method='pearson', min_periods=1)
```

```
#correl=round(correl,4)
#correl
```

Out[11]:

	0	1	2
0	1.00000	0.00051	-0.00126
1	0.00051	1.00000	0.00031
2	-0.00126	0.00031	1.00000

In [12]:
```
#Datamining: Clustering (método no supervisado de Machine Learning)
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets

from sklearn.cluster import KMeans

#Kmeans Clustering
def doKmeans(X, nclust=2):
    model = KMeans(nclust)
    model.fit(X)
    clust_labels = model.predict(X)
    cent = model.cluster_centers_
    return (clust_labels, cent)

clust_labels, cent = doKmeans(principalDf, 3)
kmeans = pd.DataFrame(clust_labels)
kmeans
```

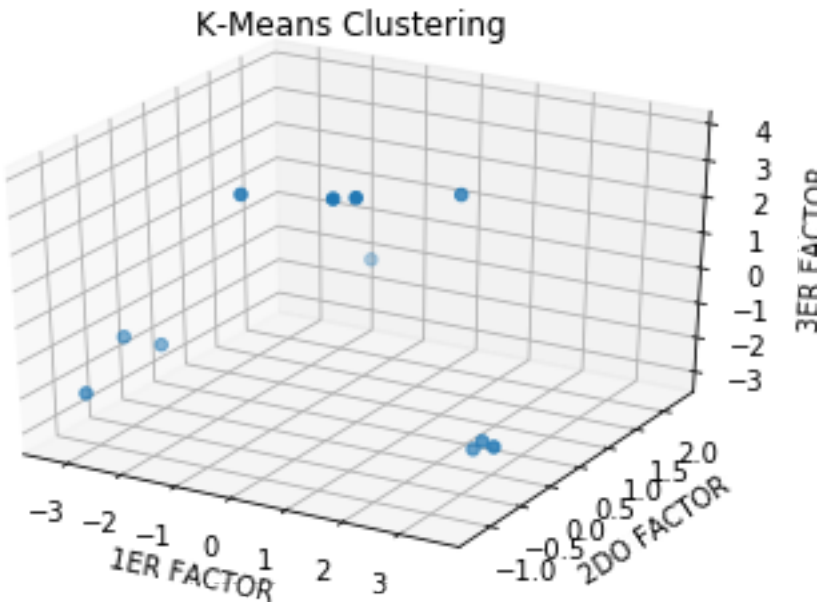
Out[12]:

	0
0	1
1	1
2	1
3	0
4	0
5	0
6	2
7	2
8	2
9	1
10	0

In [13]:
```
#Visualización de grupos de usuarios en 3D
from mpl_toolkits import mplot3d
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')

# Data for a three-dimensional line
```

```
zline = np.linspace(-5, 5, 11)
xline = np.linspace(0, 10, 11)
yline = np.linspace(0, 15, 11)
#ax.plot3D(xline, yline, zline, 'gray')

# Data for three-dimensional scattered points
zdata = principalDf[0]
xdata = principalDf[1]
ydata = principalDf[2]
#ax.scatter3D(xdata, ydata, zdata, c=zdata);
ax.scatter3D(xdata, ydata, zdata);
ax.set_title('K-Means Clustering')
ax.set_xlabel('1ER FACTOR')
ax.set_ylabel('2DO FACTOR')
ax.set_zlabel('3ER FACTOR')
```

Out[13]: Text(0.5, 0, '3ER FACTOR')

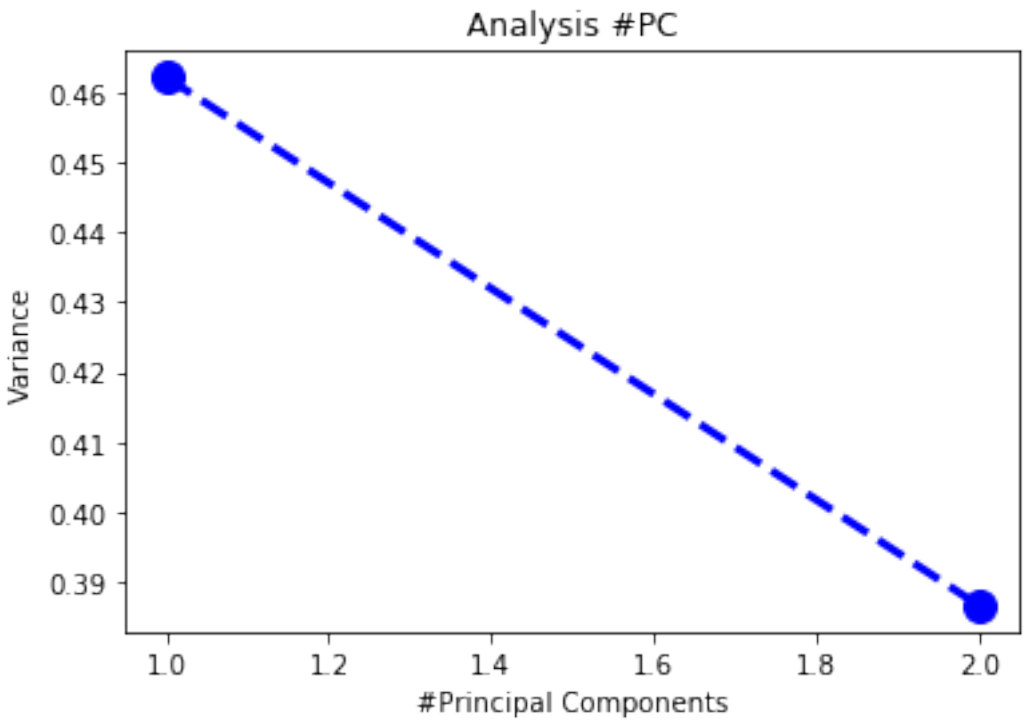


In [14]:
```
#Se aplica PCA con 2 componentes para visualizar en 2D
num_components=2
pca = PCA(num_components)
principalComponents = pca.fit_transform(x)

explained_variance_ratio_=pca.explained_variance_ratio_
explained_variance_ratio_
print('Varianzas:')
print(np.around(explained_variance_ratio_, decimals=3))
```

```
        a = range(num_components)
        num_pc= a[::1]

        principalDf = pd.DataFrame(data = principalComponents
                     , columns = num_pc)

        principalDf=round(principalDf, 2)
        print(principalDf)
```

```
Varianzas:
[0.462 0.387]
       0     1
0   3.73  0.90
1   3.83  0.72
2   2.83  1.40
3  -1.67 -2.96
4  -1.16 -1.73
5  -1.28 -3.32
6  -2.08  3.54
7  -2.52  2.95
8  -3.09  2.58
9   3.22 -1.17
10 -1.82 -2.91
```

```
In [15]: #Explicación de la varianza
        explained_variance_ratio_
        a = range(num_components)
        num_pc= np.linspace(1,num_components,num_components)

        #Visualizar en 2D
        import matplotlib.pyplot as plt

        #plt.plot(num_pc, explained_variance)
        plt.plot(num_pc, explained_variance_ratio_, color='blue', linestyle='dashed', linewid
                 marker='o', markerfacecolor='blue', markersize=12)
        #plt.ylim((0,0.4))
        plt.title('Analysis #PC')
        plt.xlabel('#Principal Components')
        plt.ylabel('Variance')
        plt.show()
```




```
In [16]: #Datamining: Clustering (método no supervisado de Machine Learning)
         clust_labels, cent = doKmeans(principalDf, 3)
         kmeans = pd.DataFrame(clust_labels)
         kmeans
```

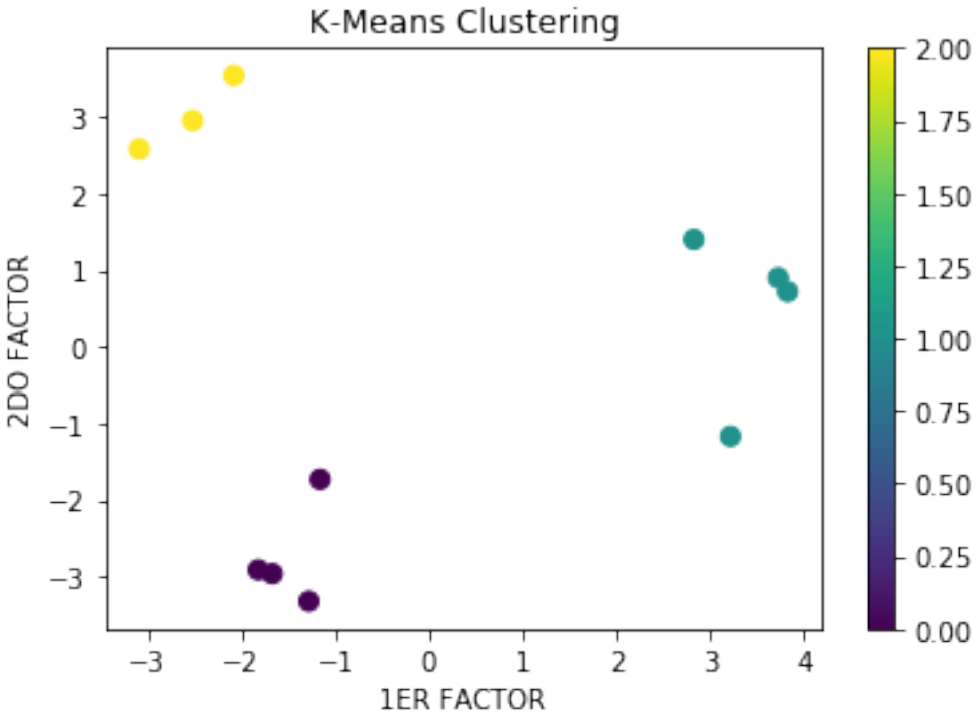
Out[16]:

	0
0	1
1	1
2	1
3	0
4	0
5	0
6	2
7	2
8	2
9	1
10	0

```
In [17]: #Visualización de grupos de usuarios en 2D
         fig = plt.figure()
         ax = fig.add_subplot(111)
         scatter = ax.scatter(principalDf[0],principalDf[1],
                              c=kmeans[0],s=50)
         ax.set_title('K-Means Clustering')
```

```
ax.set_xlabel('1ER FACTOR')
ax.set_ylabel('2DO FACTOR')
plt.colorbar(scatter)
```

Out[17]: <matplotlib.colorbar.Colorbar at 0x238cc637f28>



In [18]:
```
#Datamining: Clustering (método no supervisado de Machine Learning)
#Sin Reducir Dimensionalidad:
#Desventajas
#    a. Más costo computacional y mayor uso de memoria.
#    b. No se pueden visualizar fácilemnte los grupos de los usuarios por la alta dim
df = pd.read_csv(dataset)
df
clust_labels, cent = doKmeans(df, 3)
kmeans = pd.DataFrame(clust_labels)
kmeans
```

Out[18]:

	0
0	1
1	1
2	1
3	2
4	2
5	2

```
6   0
7   0
8   0
9   1
10  2
```

In [ ]: