+ Código + Texto

```
my_basket=[['bread','butter','wine','bananas','coffee','carrots'],
           ['tomatoes','onions','cheese','milk','potatoes'],
           ['beer','chips','asparagus','salsa','milk','apples'],
           ['olive oil','bread','butter','tomatoes','steak','carrots'],
           ['tomatoes','onions','chips','wine','ketchup','orange juice'],
           ['bread','butter','beer','chips','milk'],
           ['butter','tomatoes','carrots','coffee','sugar'],
           ['tomatoes','onions','cheese','milk','potatoes'],
           ['bread','butter','ketchup','coffee','chicken wings'],
           ['butter','beer','chips','asparagus','apples'],
           ['tomatoes','onion','beer','chips','milk','coffee']]
[all(z in i for z in ['beer','chips']) for i in my_basket]
```

```
[False, False, True, False, False, True, False, False, False, True, True]
```

```
def frecuencias(x,y):
  fx_ = sum([x in i for i in my_basket])
  fy_ = sum([y in i for i in my_basket])

  fxy_ = sum([all(z in i for z in [x,y]) for i in my_basket])

  support = fxy_/len(my_basket)
  confidence = support/(fx_/len(my_basket))
  lift = confidence/(fy_/len(my_basket))

  print("Soporte = {}".format(round(support,2)))
  print("Confianza = {}".format(round(confidence,2)))
  print("Lift = {}".format(round(lift,2)))
```

```
frecuencias('beer','cheese')
```

```
Soporte = 0.0
Confianza = 0.0
Lift = 0.0
```

```
frecuencias('onions','cheese')
```

```
Soporte = 0.18
Confianza = 0.67
Lift = 3.67
```

```
frecuencias('bread','butter')
```

```
Soporte = 0.36
Confianza = 1.0
Lift = 1.83
```

```
# Algoritmo Apriori
```

```
import numpy as np
import pandas as pd
groceries = pd.read_csv("http://dicyg.fi-c.unam.mx:8080/lalo/pypcd/presentaciones
groceries
```

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

```
transactions = list(groceries.Transaction.apply(lambda x: sorted(x.split(','))))
transactions
```

```
[['biscuit', 'bread', 'milk'],
 ['biscuit', 'bread', 'cereal', 'milk'],
 ['bread', 'tea'],
```

```
    ['bread', 'jam', 'milk'],
    ['biscuit', 'tea'],
    ['bread', 'tea'],
    ['cereal', 'tea'],
    ['biscuit', 'bread', 'tea'],
    ['bread', 'jam', 'tea'],
    ['bread', 'milk'],
    ['biscuit', 'cereal', 'coffee', 'orange'],
    ['biscuit', 'cereal', 'coffee', 'orange'],
    ['coffee', 'sugar'],
    ['bread', 'coffee', 'orange'],
    ['biscuit', 'bread', 'sugar'],
    ['cereal', 'coffee', 'sugar'],
    ['biscuit', 'bread', 'sugar'],
    ['bread', 'coffee', 'sugar'],
    ['bread', 'coffee', 'sugar'],
    ['cereal', 'coffee', 'milk', 'tea']]
```

```
1 encoder = TransactionEncoder().fit(transactions)
2 onehot = encoder.transform(transactions)
3 onehot
```

```
   array([[ True,  True, False, False, False,  True, False, False, False],
          [ True,  True,  True, False, False,  True, False, False, False],
          [False,  True, False, False, False, False, False, False,  True],
          [False,  True, False, False,  True,  True, False, False, False],
          [ True, False, False, False, False, False, False, False,  True],
          [False,  True, False, False, False, False, False, False,  True],
          [False, False,  True, False, False, False, False, False,  True],
          [ True,  True, False, False, False, False, False, False,  True],
          [False,  True, False, False,  True, False, False, False,  True],
          [False,  True, False, False, False,  True, False, False, False],
          [ True, False,  True,  True, False, False,  True, False, False],
          [ True, False,  True,  True, False, False,  True, False, False],
          [False, False, False,  True, False, False, False,  True, False],
          [False,  True, False,  True, False, False,  True, False, False],
          [ True,  True, False, False, False, False, False,  True, False],
          [False, False,  True,  True, False, False, False,  True, False],
          [ True,  True, False, False, False, False, False,  True, False],
          [False,  True, False,  True, False, False, False,  True, False],
          [False,  True, False,  True, False, False, False,  True, False],
          [False, False,  True,  True, False,  True, False, False,  True]])
```

```
1 onehot = pd.DataFrame(onehot,columns=encoder.columns_)
2
3 frequent_itemsets = apriori(onehot,min_support=0.001,max_len=3,use_colnames=True)
4 #frequent_itemsets.head(10)
5 #frequent_itemsets.tail()
```

```
1 rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
2 rules
```

```
  rules['lhs items'] = rules['antecedents'].apply(lambda x:len(x))
  rules[rules['lhs items']>1].sort_values('lift',ascending=False).head(1)
```

```
import seaborn as sns
rules['antecedents_']=rules['antecedents'].apply(lambda a: ','.join(list(a)))
rules['consequents_']=rules['consequents'].apply(lambda a: ','.join(list(a)))
pivot = rules[rules['lhs items']>1].pivot(index='antecedents_',
                                          columns='consequents_',
                                          values='lift')
#rules.head(2)
```

```
sns.heatmap(pivot,annot=True)
import matplotlib.pyplot as plt
plt.yticks(rotation=0)
```

```
plt.xticks(rotation=90)
```

```
# PCA from "scratch"
import numpy as np
import pandas as pd
```

```
df_wine=pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data'
    , header=None)
df_wine.head(1)
```

```
from sklearn.model_selection import train_test_split
X,y = df_wine.iloc[:,1:], df_wine.iloc[:,0]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=4
```

```
# Estandarización
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

```
cov_mat = np.cov(X_train_std.T)
eigen_vals,eigen_vecs = np.linalg.eig(cov_mat)
print('Eigenvals : ',eigen_vals)
```

```
Eigenvals :  [4.74376552 2.45913372 1.5276711  0.99327678 0.92313257 0.59663887
 0.46974164 0.09935613 0.17349645 0.34681782 0.23096439 0.25665489
 0.28504118]
```

```
# Gráfica de aportes de cada eigen_par
import matplotlib.pyplot as plt
tot = sum(eigen_vals)
var_exp = [ev/tot for ev in sorted(eigen_vals,reverse=True)]
cum_var_exp = np.cumsum(var_exp)
plt.bar(range(1,14),var_exp,label='varianza individual',align='center')
plt.step(range(1,14),cum_var_exp,where='mid',label='varianza acumulativa')
plt.xlabel('Índice')
plt.ylabel('Varianza')
plt.legend(loc='best')
plt.show()
```

```
  # Lista de eigen-pares
  eigen_par = [(np.abs(eigen_vals[i]), eigen_vecs[i])
               for i in range(len(eigen_vals))]
  # Ordenando de mayor a menor
  eigen_par.sort(key=lambda k: k[0], reverse=True)
```

```
eigen_par[0]
```

```
(4.743765516624778,
 array([-0.13443023, -0.49571937,  0.12605367, -0.04624624, -0.32221478,
        -0.12108437, -0.24715174,  0.02053618, -0.19383723, -0.51726172,
        -0.2924713 ,  0.31043682, -0.22611951]))
```

```
# Creamos la matriz de transformación W
w = np.hstack((eigen_par[0][1][:,np.newaxis],
               eigen_par[1][1][:,np.newaxis]))
print('Matriz W \n',w)
```

```
Matriz W
 [[-0.13443023  0.25680248]
 [-0.49571937 -0.21988534]
 [ 0.12605367 -0.08398481]
```

```
    [-0.04624624 -0.56428416]
    [-0.32221478  0.18146701]
    [-0.12108437  0.16951285]
    [-0.24715174 -0.60464068]
    [ 0.02053618 -0.10406031]
    [-0.19383723  0.08039695]
    [-0.51726172  0.01884124]
    [-0.2924713   0.2717176 ]
    [ 0.31043682 -0.20011018]
    [-0.22611951 -0.038156  ]]
```

```
print('Original : ',X_train_std[0])
print('Proyectada : ',X_train_std[0].dot(w))
```

```
Original :  [ 0.62844732  1.08120605 -0.65212742  0.         -0.8414766  -1.0033
 -1.51706225  1.71144809 -1.23077056  0.33317435 -0.64137827 -1.07090115
 -0.51821917]
Proyectada :  [0.13860601 0.36194062]
```

```
X_train_pca = X_train_std.dot(w)
X_train_pca.shape
```

```
(124, 2)
```

```
# SciKitLearn :)
```

```
# Bibliotecas
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
df_wine=pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data'
    , header=None)
```

```
from sklearn.model_selection import train_test_split
X,y = df_wine.iloc[:,1:].values, df_wine.iloc[:,0].values
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=4
```

```
#y_train
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

```
# Ayuda con regresión logística
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.decomposition import PCA
```

```
  pca = PCA(n_components=2)
  lr = LogisticRegression(multi_class='auto',solver='liblinear')

  X_train_pca = pca.fit_transform(X_train_std)
  X_test_pca = pca.transform(X_test_std)
  lr.fit(X_train_pca,y_train)
```

```
LogisticRegression(solver='liblinear')
```

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_train_pca,y_train,clf=lr,legend=2)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()
```

```
plot_decision_regions(X_test_pca,y_test,clf=lr,legend=2)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()
```

```
# LDA CON PYTHON
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train_std, y_train)
```

```
lr = LogisticRegression()
lr.fit(X_train_lda, y_train)
```

```
LogisticRegression()
```

```
 plot_decision_regions(X_train_lda, y_train, clf=lr, legend=2)
 plt.xlabel('LD 1')
 plt.ylabel('LD 2')
 plt.show()
```

```
X_test_lda = lda.transform(X_test_std)
plot_decision_regions(X_test_lda, y_test, clf=lr, legend=2)
plt.xlabel('LD 1')
plt.ylabel('LD 2')
plt.show()
```

1