امیرحسین رئیس محمدی ۹۷۰۹۶۵۴

۱.

۱,۱.

```
In [23]: from sklearn.datasets import load_breast_cancer
         BreastCancer = load_breast_cancer()
         x=BreastCancer.DESCR
         print (x)
```

```
Breast Cancer Wisconsin (Diagnostic) Database
=============================================

Notes
-----
Data Set Characteristics:
uble click to hide of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign
```

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

References

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

۱,۲.

```
In [44]: x=BreastCancer.feature_names
         print (x)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

```
In [45]: x=BreastCancer.data
         print (x)
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
```

```
In [46]: x=BreastCancer.keys()
         print (x)
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

۱,۳.

```
In [48]: import pandas as pd
         import numpy as np
         df = pd.DataFrame(BreastCancer.data, columns=BreastCancer.feature_names)
```

۱,۴.

```
In [49]: df.describe()
```

Out[49]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	16.269190	25.677223
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242	6.146258
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000	12.020000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000	21.080000
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000	25.410000
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000	29.720000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	36.040000	49.540000

8 rows × 30 columns

۱,۵.

```
In [24]: df['target'] = BreastCancer.target
         df.set_index('target',inplace=True)
```

.۱,۶

```
In [26]: df.index.value_counts()
Out[26]: 1    357
         0    212
         Name: target, dtype: int64
```

```
In [27]: x=BreastCancer.target_names
         print (x)

         ['malignant' 'benign']
```

همانطور که مشاهده میکنید در دیتاست تعداد مقادیر malignant ۲۱۲ و benign ۳۵۷ است.

.۱,۷

```
In [45]: from sklearn.model_selection import train_test_split
         X = df[BreastCancer['feature_names']]
         y = df.index
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

.۱,۸

```
In [56]: X_train.shape
Out[56]: (426, 30)

In [57]: X_test.shape
Out[57]: (143, 30)

In [58]: y_train.shape
Out[58]: (426,)

In [59]: y_test.shape
Out[59]: (143,)
```

.۱,۹

```
In [60]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors = 6)
         knn.fit(X_train, y_train)
         meanAccuracy = knn.score(X_test, y_test)
         meanAccuracy

Out[60]: 0.9230769230769231
```

.۱,۱۰

```
In [61]: ansTest = knn.predict(X_test)
         ansTest

Out[61]: array([0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1,
                0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
                0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
                0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0,
                1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0])
```

.۱,۱۱

این تابع با توجه به X_train ، مقادیر X_test را پیشبینی میکند.

.۱,۱۲

```
In [62]: from sklearn.preprocessing import MinMaxScaler
         minmax = MinMaxScaler()
         minmax.fit(X)
         normalized_Xtrain = minmax.transform(X_train)
         normalized_Xtest = minmax.transform(X_test)
```

.۱,۱۳

```
In [64]: knnn = KNeighborsClassifier(n_neighbors = 6)
         knnn.fit(normalized_Xtrain,y_train)

Out[64]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=6, p=2,
                    weights='uniform')
```

.۱,۱۴

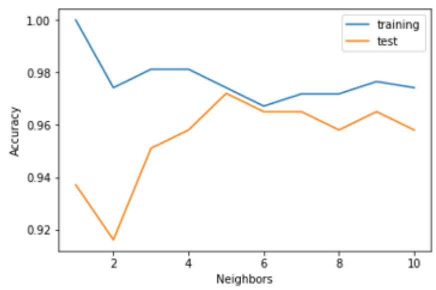
```
In [67]: meanAccuracy_train = knnn.score(normalized_Xtrain, y_train)
         meanAccuracy_test = knnn.score(normalized_Xtest, y_test)
         print (meanAccuracy_train)
         print (meanAccuracy_test)

            0.9671361502347418
            0.965034965034965
```

.۱,۱۵

```
In [71]: training_accuracy = []
         test_accuracy = []

         neighbors = range (1, 11)

         for i in neighbors:
             knnnn = KNeighborsClassifier(n_neighbors = i)
             knnnn.fit(normalized_Xtrain,y_train)
             training_accuracy.append(knnnn.score(normalized_Xtrain, y_train))
             test_accuracy.append(knnnn.score(normalized_Xtest, y_test))
```

۱,۱۶.

```
In [43]: import matplotlib.pyplot as plt

         plt.plot(neighbors, training_accuracy, label='training')
         plt.plot(neighbors, test_accuracy, label='test')
         plt.ylabel('Accuracy')
         plt.xlabel('Neighbors')
         plt.legend()

Out[43]: <matplotlib.legend.Legend at 0x1d74a636f60>
```



۱,۱۷.

۲.

۲,۱. دیتاست دانلود شد.

۲,۲.

```
In [1]: import numpy as np
        import pandas as pd
        df = pd.read_csv('dataset_54_vehicle.csv', sep=',')
        df.head()

Out[1]:
```

	COMPACTNESS	CIRCULARITY	DISTANCE_CIRCULARITY	RADIUS_RATIO	PR.AXIS_ASPECT_RATIO	MAX.LENGTH_ASPECT_RATIO	SCATTER_RATIO	ELONGA
0	95	48	83	178	72	10	162	
1	91	41	84	141	57	9	149	
2	104	50	106	209	66	10	207	
3	93	41	82	159	63	9	144	
4	85	44	70	205	103	52	149	

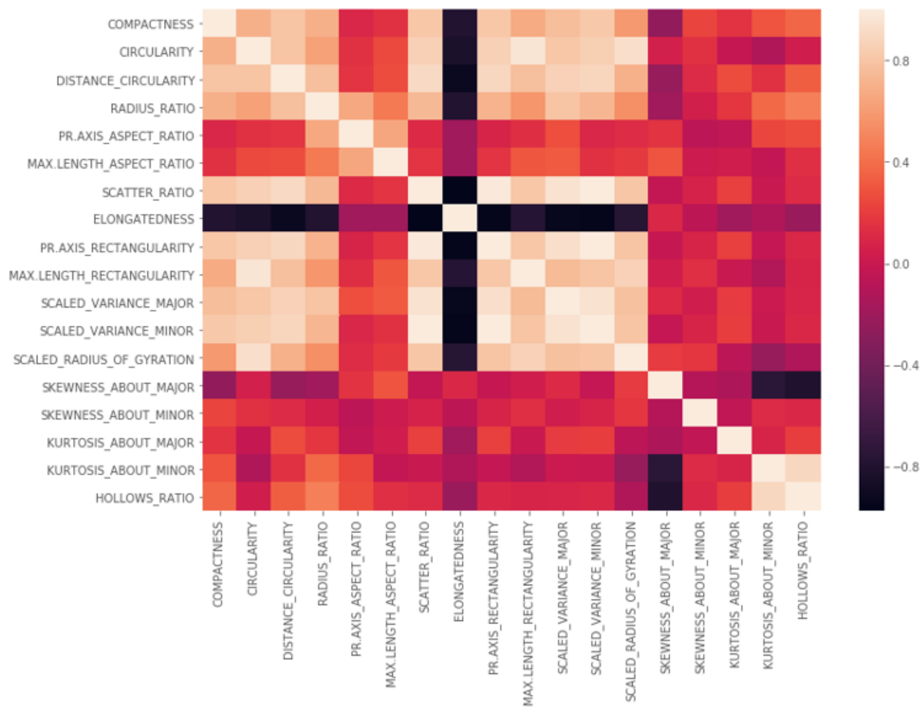
۲,۳.

```
In [2]: variables = df['Class'].unique()
        print (variables)

    ['van' 'saab' 'bus' 'opel']
```

۲,۴.

```
In [6]: from sklearn import preprocessing
        import matplotlib
        import matplotlib.pyplot as plt
        import seaborn as sns
        plt.figure(figsize=(12,8))
        sns.heatmap(df.corr())
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x29ba632eb70>
```



۲,۵.

```
In [20]: x = df.iloc[:,:-1]
         y = df.iloc[:,18]
```

۲,۶.

```
In [21]: 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_state=42)
```

۲,۷.

```
In [22]: from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score
         from sklearn import tree


         model = tree.DecisionTreeClassifier(criterion='entropy', max_depth=5, max_features=4)
```

.۲,۸

```
In [23]: import random
         from scipy.stats import randint
         params = {"max_depth": [3, None],
                   "max_features": randint(1, 9),
                   "min_samples_leaf": randint(1, 9)}
         print(params)

         {'max_depth': [3, None], 'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x00000152BBE796D8>, 'min_samp
         les_leaf': <scipy.stats._distn_infrastructure.rv_frozen object at 0x00000152BBDA53C8>}
```

.۲,۹

```
In [27]: from sklearn.model_selection import RandomizedSearchCV
         tree = DecisionTreeClassifier()
         tree_cv = RandomizedSearchCV(tree, params, cv=5)
         tree_cv.fit(x ,y)

Out[27]: RandomizedSearchCV(cv=5, error_score='raise',
                   estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best'),
                   fit_params=None, iid=True, n_iter=10, n_jobs=1,
                   param_distributions={'max_depth': [3, None], 'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0
         x00000152BBE796D8>, 'min_samples_leaf': <scipy.stats._distn_infrastructure.rv_frozen object at 0x00000152BBDA53C8>},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score='warn', scoring=None, verbose=0)
```

.۲,۱۰

```
In [28]: print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
         print("Best score is {}".format(tree_cv.best_score_))

         Tuned Decision Tree Parameters: {'max_depth': None, 'max_features': 7, 'min_samples_leaf': 1}
         Best score is 0.6843971631205674
```

.۲,۱۱

```
In [36]: print("Accuracy is ", accuracy_score(y_test,y_pred)*100)

         Accuracy is  93.52941176470588
```

.۲,۱۲

مقدار CV هرچه بیشتر باشد مدل بهتر آموزش میبیند.

.۲,۱۳

```
In [35]: from sklearn import tree
model = tree.DecisionTreeClassifier( max_depth = None, max_features= 5,min_samples_leaf= 3)
model.fit(x, y)
y_pred=model.predict(x_test)
y_pred
```

```
Out[35]: array(['bus', 'van', 'bus', 'van', 'bus', 'van', 'van', 'saab', 'bus',
       'saab', 'saab', 'bus', 'van', 'saab', 'opel', 'van', 'saab', 'bus',
       'opel', 'opel', 'van', 'van', 'saab', 'opel', 'opel', 'saab',
       'opel', 'van', 'van', 'van', 'opel', 'van', 'bus', 'van', 'van',
       'bus', 'bus', 'saab', 'bus', 'saab', 'van', 'bus', 'saab', 'van',
       'opel', 'saab', 'bus', 'van', 'bus', 'van', 'bus', 'bus', 'saab',
       'opel', 'bus', 'opel', 'saab', 'opel', 'saab', 'saab', 'bus',
       'van', 'saab', 'opel', 'bus', 'bus', 'saab', 'saab', 'opel',
       'opel', 'saab', 'bus', 'saab', 'bus', 'van', 'van', 'saab', 'opel',
       'opel', 'van', 'bus', 'van', 'saab', 'opel', 'saab', 'opel',
       'saab', 'bus', 'van', 'saab', 'saab', 'opel', 'bus', 'bus', 'opel',
       'saab', 'van', 'van', 'bus', 'opel', 'saab', 'saab', 'opel', 'van',
       'bus', 'bus', 'saab', 'van', 'opel', 'saab', 'van', 'bus', 'van',
       'bus', 'bus', 'van', 'bus', 'opel', 'bus', 'saab', 'bus', 'opel',
       'saab', 'van', 'bus', 'saab', 'bus', 'van', 'bus', 'opel', 'bus',
       'van', 'bus', 'saab', 'bus', 'bus', 'bus', 'saab', 'saab', 'bus',
       'van', 'bus', 'bus', 'bus', 'van', 'opel', 'saab', 'opel', 'saab',
       'opel', 'saab', 'bus', 'bus', 'opel', 'van', 'van', 'bus', 'van',
       'saab', 'bus', 'bus', 'van', 'opel', 'saab', 'opel', 'opel',
       'saab', 'opel', 'saab', 'van'], dtype=object)
```

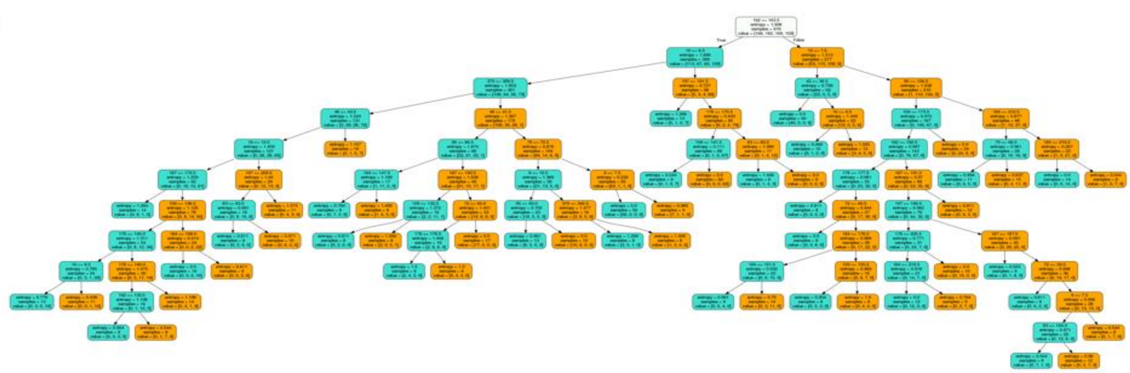
.۲,۱۴

```
In [34]: from sklearn.tree import export_graphviz
export_graphviz(model,
                out_file='dot_data.dot',
                feature_names = data.columns,
                class_names = 'Class',
                rounded = True, proportion = False,
                precision = 2, filled = True)
```

.۲,۱۵

```
In [41]: try:
    from StringIO import StringIO
except ImportError:
    from io import StringIO
from sklearn import tree
import pydotplus

dotfile = StringIO()
tree.export_graphviz(model,
                out_file=dotfile,
                feature_names = x.columns,
                class_names = 'Class',
                rounded = True, proportion = False,
                precision = 2, filled = True)
graph=pydotplus.graph_from_dot_data(dotfile.getvalue())
graph.write_png("dtree.png")
Image('tree.png')
```

Out[14]:



.۲,۱۶

```
In [16]: graph.write_pdf("dtree.pdf")
```

.۳

.۳,۱

```
In [7]: from sklearn import datasets

        iris = datasets.load_iris()
        iris
```

```
Out[7]: {'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],
                [5. , 3.6, 1.4, 0.2],
                [5.4, 3.9, 1.7, 0.4],
                [4.6, 3.4, 1.4, 0.3],
                [5. , 3.4, 1.5, 0.2],
                [4.4, 2.9, 1.4, 0.2],
                [4.9, 3.1, 1.5, 0.1],
                [5.4, 3.7, 1.5, 0.2],
                [4.8, 3.4, 1.6, 0.2],
                [4.8, 3. , 1.4, 0.1],
                [4.3, 3. , 1.1, 0.1],
                [5.8, 4. , 1.2, 0.2],
                [5.7, 4.4, 1.5, 0.4],
                [5.4, 3.9, 1.3, 0.4],
                [5.1, 3.5, 1.4, 0.3],
                [5.7, 3.8, 1.7, 0.3],
```

.۳,۲

```
In [12]: from sklearn.cluster import KMeans
         samples = df.iloc[:,:4]
         model = KMeans(n_clusters=3)
         model.fit(samples)
```

```
Out[12]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
             n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
             random_state=None, tol=0.0001, verbose=0)
```

```
In [13]: labels = model.predict(samples)
         labels
```

```
Out[13]: array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1])
```

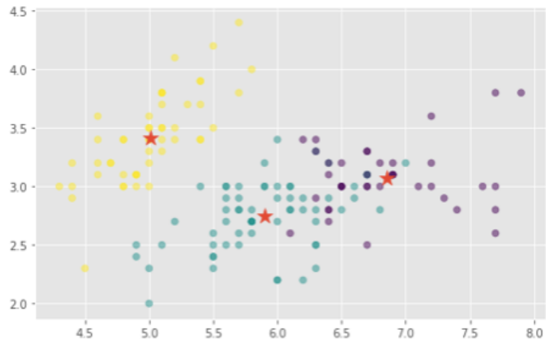
.۳,۳

```
In [14]: centroids = model.cluster_centers_
         centroids
```

```
Out[14]: array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
                [5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
                [5.006     , 3.418     , 1.464     , 0.244     ]])
```

.۳,۴

In [18]:
```
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
matplotlib.style.use('ggplot')
plt.figure(figsize=(8,5))

xs = samples.iloc[:,0]
ys = samples.iloc[:,1]

plt.scatter(xs, ys, c=labels, alpha=0.5)

centroids_x = centroids[:,0]
centroids_y = centroids[:,1]

plt.scatter(centroids_x, centroids_y, marker='*', s=200)
plt.show()
```



.۳,۵

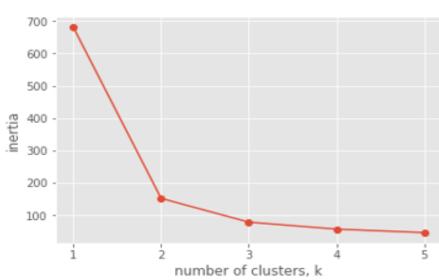
In [19]:
```
print(model.inertia_)
```

```
78.94084142614602
```

.۳,۶

In [20]:
```
ks = range(1, 6)
inertias = []

for k in ks:
    model = KMeans(n_clusters=k)
    model.fit(df.iloc[:,:4])
    inertias.append(model.inertia_)

print(inertias)

# Plot ks vs inertias
plt.plot(ks, inertias, '-o')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

```
[680.8244, 152.36870647733906, 78.94084142614602, 57.31787321428571, 46.53558205128205]
```



.۳,۷

با توجه به اینکه معیار اصلی ما برای کیفیت کلاسترینگ استفاده از شاخص inertiaمیباشد، پس در ابتدا با تعداد یک کلاستر بیشترین خطا برای مدل بدست می‌آید زیرا بیشترین فاصله از مرکز یک کلاستر برای کل دیتاست به وجود می‌آید. هر چه تعداد کلاسترها بیشتر شود مقدار inertia کمتر شده و دقت کلاسترهای بدست آمده بیشتر میشود ولی در طرف مقابل عمومیت مدل از دست خواهد رفت به همین دلیل باید تعداد کلاسترهایی را انتخاب کرد که مقدارهای بعد از آن مقدار ییت به صورت محسوسی تغییر نخواهد کرد. به طور مثال در این سوال تعداد سه کلاستر برای این دیتاست مناسب است زیر تا قبل از آن شاخص inertia به صورت نمایی کاهش می‌یابد و پس از آن تغییرات محسوسی بروی کاهش شاخص inertiaبدست نمی‌آید.

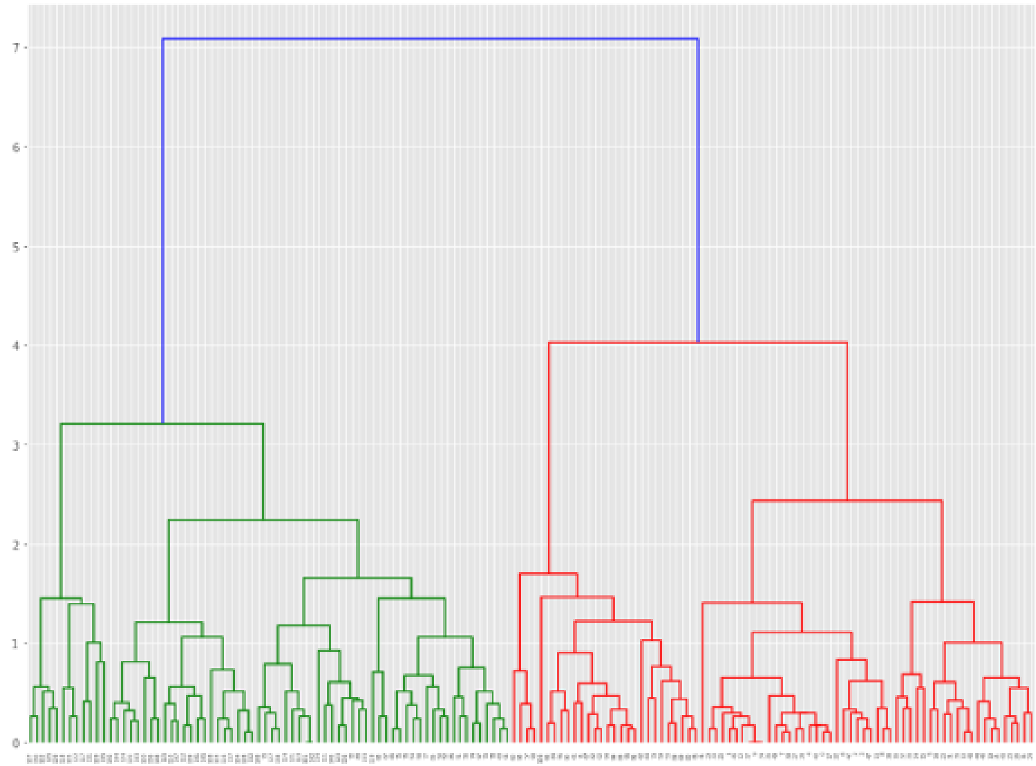
.۴

.۴,۱ و ۴,۲

```
In [1]: from sklearn import datasets

        iris = datasets.load_iris()
        iris
```

```
Out[1]: {'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],
                [5. , 3.6, 1.4, 0.2],
                [5.4, 3.9, 1.7, 0.4],
                [4.6, 3.4, 1.4, 0.3],
                [5. , 3.4, 1.5, 0.2],
                [4.4, 2.9, 1.4, 0.2],
                [4.9, 3.1, 1.5, 0.1],
                [5.4, 3.7, 1.5, 0.2],
                [4.8, 3.4, 1.6, 0.2],
                [4.8, 3. , 1.4, 0.1],
                [4.3, 3. , 1.1, 0.1],
                [5.8, 4. , 1.2, 0.2],
                [5.7, 4.4, 1.5, 0.4],
                [5.4, 3.9, 1.3, 0.4],
                [5.1, 3.5, 1.4, 0.3],
                [5.7, 3.8, 1.7, 0.3],
```

In [4]:
```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
matplotlib.style.use('ggplot')

linkage_matrix = linkage(iris.data, 'complete')

plt.figure(figsize=(16,12))
dendrogram(linkage_matrix)
plt.show()
```



.۴,۳

In [5]:
```
clusters = fcluster(linkage_matrix, 6, criterion="distance")
clusters
```

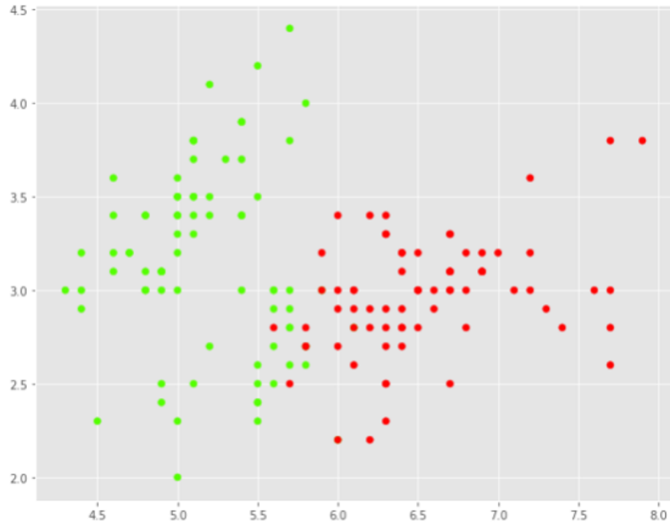
Out[5]:
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 1,
       2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 1, 1, 1,
       2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

In [6]:
```
clusters = fcluster(linkage_matrix, 2, criterion='maxclust')
clusters
```

Out[6]:
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 1,
       2, 2, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 1, 1, 1,
       2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

۴,۴.

```
In [7]: plt.figure(figsize=(10, 8))
        plt.scatter(iris.data[:,0], iris.data[:,1], c=clusters, cmap='prism')
        plt.show()
```



۵.

۵,۱.

```
In [2]: from sklearn.datasets import load_boston
        import numpy as np
        import pandas as pd

        boston_dataset = load_boston()
        boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
        boston.head()
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

.۵,۲

```
In [3]: boston['Price'] = boston_dataset.target
        boston.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Price
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

.۵,۳

```
In [4]: from sklearn.linear_model import LinearRegression
        x= boston[["CRIM","ZN"]]
        y= boston[["Price"]]
        model=LinearRegression()
        model = LinearRegression().fit(x, y)
        r_sq = model.score(x, y)
        print('coefficient of determination:', r_sq)
        print('intercept:', model.intercept_)
        print('slope:', model.coef_)

coefficient of determination: 0.23256130554722754
intercept: [22.46681692]
slope: [[-0.34977589  0.11642402]]
```

.۵,۴

```
In [21]: from sklearn.model_selection import train_test_split
         X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3, random_state=5)
```

.۵,۵

.۵,۵,۱ و ۵,۵,۲ و ۵,۵,۳

```
In [62]: from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error
         from sklearn.metrics import r2_score

         lin_model = LinearRegression()
         lin_model.fit(X_train, Y_train)
         y_train_predict = lin_model.predict(X_train)
         y_test_predict = lin_model.predict(X_test)
```

.۵,۶

```
In [63]: mse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
         r2 = r2_score(Y_train, y_train_predict)

         print("The model performance for training set")
         print("--------------------------------------")
         print('MSE is {}'.format(mse))
         print('R2 score is {}'.format(r2))
         print("\n")

         mse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
         r2 = r2_score(Y_test, y_test_predict)

         print("The model performance for testing set")
         print("--------------------------------------")
         print('MSE is {}'.format(mse))
         print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
MSE is 7.6560938362551285
R2 score is 0.26580934567509706


The model performance for testing set
--------------------------------------
MSE is 8.918591673933312
R2 score is 0.16349145122401265
```

۵,۷.

```
In [64]: x= boston[["LSTAT"]]
         y= boston[["Price"]]
         model=LinearRegression()
         model = LinearRegression().fit(x, y)
         r_sq = model.score(x, y)
         print('coefficient of determination:', r_sq)
         print('intercept:', model.intercept_)
         print('slope:', model.coef_)
```

```
coefficient of determination: 0.5441462975864799
intercept: [34.55384088]
slope: [[-0.95004935]]
```

```
In [65]: X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3, random_state=5)
```

```
In [66]: lin_model = LinearRegression()
         lin_model.fit(X_train, Y_train)
         y_train_predict = lin_model.predict(X_train)
         y_test_predict = lin_model.predict(X_test)
```

```
In [67]: mse = (np.sqrt(mean_squared_error(Y_train, y_train_predict)))
         r2 = r2_score(Y_train, y_train_predict)

         print("The model performance for training set")
         print("--------------------------------------")
         print('MSE is {}'.format(mse))
         print('R2 score is {}'.format(r2))
         print("\n")

         mse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
         r2 = r2_score(Y_test, y_test_predict)

         print("The model performance for testing set")
         print("--------------------------------------")
         print('MSE is {}'.format(mse))
         print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
MSE is 5.942398232895452
R2 score is 0.5576990599447106


The model performance for testing set
--------------------------------------
MSE is 6.777234336301447
R2 score is 0.5169602987600737
```

۵,۸.

مقدار mse کاهش یافته است ، طبیعتا اگه پارامتری که به مدل اضافه میکنیم به متغیر هدفمون مربوط باشه دقت مدل افزایش میابد.

۵,۹.

۶.

۶,۱.

```
In [2]: from sklearn.datasets import load_breast_cancer
        import pandas as pd
        import numpy as np
        Cancer=load_breast_cancer()
        Cancer
```

```
Out[2]: {'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
                 1.189e-01],
                [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
                 8.902e-02],
                [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
                 8.758e-02],
                ...,
                [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
                 7.820e-02],
                [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
                 1.240e-01],
                [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
                 7.039e-02]]),
         'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
```

۶,۲.

```
In [3]: from sklearn.model_selection import train_test_split
        df = pd.DataFrame(Cancer.data, columns=Cancer.feature_names)
        df['target'] = Cancer.target
        X = df[Cancer['feature_names']]
        y = df['target']
        X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = 0.2, random_state=0)

        from sklearn.neighbors import KNeighborsClassifier
        knn = KNeighborsClassifier(n_neighbors = 8)
        knn.fit(X_train, y_train)
        y_pred=knn.predict(X_test)
        print(y_pred)
```

```
[0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 1 1 0 1 1 1 1 0 1 0 1 0 1 0 1
 0 1 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 1 1
 0 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1
 0 0 1]
```

۶,۳.

```
In [4]: from sklearn.metrics import confusion_matrix
        from sklearn.metrics import classification_report
```

۶,۴.

```
In [5]: Confiuse=confusion_matrix(y_test, y_pred)
        print (Confiuse)
```

```
[[44  3]
 [ 3 64]]
```

۴۴ تا به درستی پیش بینی شده که benign هستند. ۶۴ تا به درستی پیش بینی شده که malignant هستند . ۳ تا به اشتباه پیش بینی شده که malignant هستند . و ۳ تا هم به اشتباه پیشبینی شده است که benign هستند.

۶,۵.

```
In [46]: Report=classification_report (y_test, y_pred)
         print (Report)

                 precision    recall  f1-score   support

              0       0.94      0.94      0.94        47
              1       0.96      0.96      0.96        67

    avg / total       0.95      0.95      0.95       114
```

مقادیر Percision و Recall و Support و Score را برای مقادیر صفر و یک به دست آورده است.

۶,۶.

```
In [47]: from sklearn.preprocessing import normalize
         Normal=normalize(Confiuse, norm='l1')
         print(Normal)

  [[0.93617021 0.06382979]
   [0.04477612 0.95522388]]
```

۶,۷.

```
In [48]: dff = pd.DataFrame(Normal, index=['benign','malignant'],columns=['benign','malignant'])
         print (dff)

                 benign  malignant
    benign     0.936170   0.063830
    malignant  0.044776   0.955224
```

۶,۸.

نمودار های زیر نشان دهنده دقت بدست آمده مدل هستند. و هرچه نمودار از خط Y=X دور تر باشد دقت بهتر است ، به عنوان مثال در این نمودار ها سمت چپ بالا بالا ترین دقت ممکن را به دست آورده است. نمودار سمت چپ پایین یک دقت خوب به دست آورده و سمت راست پایین دقت بدی را به دست آورده است.

۶,۹.

```
In [51]: y_pred_prob=knn.predict_proba(X_test)
         print (y_pred_prob)
```

```
[[0.75  0.25 ]
 [0.    1.   ]
 [0.    1.   ]
 [0.5   0.5  ]
 [0.    1.   ]
 [0.    1.   ]
 [0.    1.   ]
 [0.    1.   ]
 [0.    1.   ]
 [0.    1.   ]
 [0.375 0.625]
 [0.125 0.875]
 [0.    1.   ]
 [0.625 0.375]
 [0.375 0.625]
 [1.    0.   ]
 [0.    1.   ]
 [1.    0.   ]
 [1.    0.   ]
```

۶,۱۰.

۶,۱۱.

۶,۱۲.

۷.

۷,۱.

```
In [2]: import pandas as pd
        import numpy as np

        df=pd.read_excel("Online Retail.xlsx")
        df.head()
```

Out[2]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

۷,۲.

نصب پکیج انجام شد.

۷,۳.

```
In [4]: from mlxtend.frequent_patterns import apriori,association_rules
```

.۷,۴

```
In [6]: df["Description"]=df["Description"].str.strip()
```

.۷,۵

```
In [7]: print("Orginal Size : " + str(df.size))
        df["InvoiceNo"].replace('', np.nan, inplace=True)
        df.dropna(subset=['InvoiceNo'], inplace=True)
        print("Reduced Size : " + str(df.size))

        df["InvoiceNo"]=df["InvoiceNo"].astype("str")

        Orginal Size : 4335272
        Reduced Size : 4335272
```

.۷,۶

```
In [8]: df=df[~df.InvoiceNo.str.contains("C")]
```

.۷,۷

```
In [35]: basket = (df[df['Country'] =="France"]
         .groupby(['InvoiceNo', 'Description'])['Quantity']
         .sum().unstack().reset_index().fillna(0).set_index('InvoiceNo'))
         basket.head()
```

Out[35]:

Description	10 COLOUR SPACEBOY PEN	12 COLOURED PARTY BALLOONS	12 EGG HOUSE PAINTED WOOD	12 MESSAGE CARDS WITH ENVELOPES	12 PENCIL SMALL TUBE WOODLAND	12 PENCILS SMALL TUBE RED RETROSPOT	12 PENCILS SMALL TUBE SKULL	12 PENCILS TALL TUBE POSY	12 PENCILS TALL TUBE RED RETROSPOT	12 PENCILS TALL TUBE WOODLAND	...	WRAP VINTAGE PETALS DESIGN	YELLO COA RAC PAR FASHIO
InvoiceNo													
536370	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0
536852	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0
536974	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0
537065	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0
537463	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0

5 rows × 1563 columns

.۷,۸

In [36]:
```python
basket=basket.applymap(lambda x: 1 if x > 0 else 0)
basket.head()
```

Out[36]:

Description	10 COLOUR SPACEBOY PEN	12 COLOURED PARTY BALLOONS	12 EGG HOUSE PAINTED WOOD	12 MESSAGE CARDS WITH ENVELOPES	12 PENCIL SMALL TUBE WOODLAND	12 PENCILS SMALL TUBE RED RETROSPOT	12 PENCILS SMALL TUBE SKULL	12 PENCILS TALL TUBE POSY	12 PENCILS TALL TUBE RED RETROSPOT	12 PENCILS TALL TUBE WOODLAND	...	WRAP VINTAGE PETALS DESIGN	YELLO COA RAC PAR FASHIO
InvoiceNo													
536370	0	0	0	0	0	0	0	0	0	0	...	0	
536852	0	0	0	0	0	0	0	0	0	0	...	0	
536974	0	0	0	0	0	0	0	0	0	0	...	0	
537065	0	0	0	0	0	0	0	0	0	0	...	0	
537463	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 1563 columns

۷,۹.

In [37]:
```python
basket=basket.drop("POSTAGE",axis=1)
```

۷,۱۰.

In [42]:
```python
frequent_itemsets = apriori(basket, min_support=0.07, use_colnames=True)
frequent_itemsets.head()
```

Out[42]:

	support	itemsets
0	0.071429	(4 TRADITIONAL SPINNING TOPS)
1	0.096939	(ALARM CLOCK BAKELIKE GREEN)
2	0.102041	(ALARM CLOCK BAKELIKE PINK)
3	0.094388	(ALARM CLOCK BAKELIKE RED)
4	0.081633	(BAKING SET 9 PIECE RETROSPOT)

۷,۱۱.

In [43]:
```python
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.head()
```

Out[43]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(ALARM CLOCK BAKELIKE PINK)	(ALARM CLOCK BAKELIKE GREEN)	0.102041	0.096939	0.073980	0.725000	7.478947	0.064088	3.283859
1	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE PINK)	0.096939	0.102041	0.073980	0.763158	7.478947	0.064088	3.791383
2	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE GREEN)	0.094388	0.096939	0.079082	0.837838	8.642959	0.069932	5.568878
3	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.096939	0.094388	0.079082	0.815789	8.642959	0.069932	4.916181
4	(ALARM CLOCK BAKELIKE PINK)	(ALARM CLOCK BAKELIKE RED)	0.102041	0.094388	0.073980	0.725000	7.681081	0.064348	3.293135

۷,۱۲.

In [44]:
```
rules[ (rules['lift'] >= 6) &
       (rules['confidence'] >= 0.8) ]
```

Out[44]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
2	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE GREEN)	0.094388	0.096939	0.079082	0.837838	8.642959	0.069932	5.568878
3	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.096939	0.094388	0.079082	0.815789	8.642959	0.069932	4.916181
16	(SET/6 RED SPOTTY PAPER PLATES)	(SET/20 RED RETROSPOT PAPER NAPKINS)	0.127551	0.132653	0.102041	0.800000	6.030769	0.085121	4.336735
18	(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS)	0.127551	0.137755	0.122449	0.960000	6.968889	0.104878	21.556122
19	(SET/6 RED SPOTTY PAPER CUPS)	(SET/6 RED SPOTTY PAPER PLATES)	0.137755	0.127551	0.122449	0.888889	6.968889	0.104878	7.852041
20	(SET/6 RED SPOTTY PAPER PLATES, SET/6 RED SPOT...	(SET/20 RED RETROSPOT PAPER NAPKINS)	0.122449	0.132653	0.099490	0.812500	6.125000	0.083247	4.625850
21	(SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...	(SET/6 RED SPOTTY PAPER CUPS)	0.102041	0.137755	0.099490	0.975000	7.077778	0.085433	34.489796
22	(SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...	(SET/6 RED SPOTTY PAPER PLATES)	0.102041	0.127551	0.099490	0.975000	7.644000	0.086474	34.897959

۷,۱۳.

در تصویر بالا مشاهده میکنید مقادیر ستون دو اگر خریداری شوند با توجه به مقادیر عددی همان سطر احتمال خرید وجود دارد.