به نام خدا

پاسخ تکلیف دوم درس داده کاوی

نام دانشجو :

فردین ابوالفتحی

9703724

سوال 1 :

k- نزدیک‌ترین همسایگی (k-Nearest Neighbors) یک روش ناپارامتری است که در داده‌کاوی، یادگیری ماشین و تشخیص الگو مورد استفاده قرار می‌گیرد. بر اساس آمارهای ارائه شده در وب‌سایت kdnuggets الگوریتم k-نزدیک‌ترین همسایگی یکی از ده الگوریتمی است که بیشترین استفاده را در پروژه‌های گوناگون یادگیری ماشین و داده‌کاوی، هم در صنعت و هم در دانشگاه داشته است.

یکی از دلایل اصلی پرکاربرد بودن الگوریتم‌های طبقه‌بندی (Classification) آن است که «تصمیم‌گیری» یکی از چالش‌های اساسی موجود در اغلب پروژه‌های تحلیلی است. برای مثال، تصمیم‌گیری درباره اینکه آیا مشتری X پتانسیل لازم برای مورد هدف قرار داده شدن در کارزارهای دیجیتال یک کسب‌وکار را دارد یا خیر و یا اینکه آیا یک مشتری وفادار است یا نه از جمله مسائل تصمیم‌گیری به حساب می‌آیند که در فرآیند تحلیل قصد پاسخ‌دهی به آن‌ها وجود دارد. نتایج این تحلیل‌ها بسیار تأمل‌برانگیز هستند و به‌طور مستقیم به پیاده‌سازی نقشه راه در یک سازمان یا کسب‌وکار کمک می‌کنند.

1.1

متد DESCR یک توضیح کلی درباره دیتاست مربوطه:

```
cancer = load_breast_cancer()
print(cancer.DESCR)
```

```
:Attribute Information:
    ================================== ====== ======
                                        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
```

1.2

()Keysلیستی از تمام کلید های موجود در فرهنگ لغت را باز می گرداند.

()Feature_names تمام feature هارا لیست میکند.

```
print(cancer.keys())
print(cancer.feature_names)
print(cancer.data)
```

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

1.3

تبدیل داده ها به فرمت Pandas DataFrame

```
df = pd.DataFrame(np.c_[cancer['data'], cancer['target']],
                  columns= np.append(cancer['feature_names'], ['target']))
```

1.4

```
print(cancer.DESCR)
```

```
    :Number 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.
```

1.5

```
df = pd.DataFrame(np.c_[cancer['data'], cancer['target']],
                  columns= np.append(cancer['feature_names'], ['target']))
```

1.6

تابع () Index.value_counts یک شیء حاوی شمارهٔ ارزش های منحصر به فرد را بازمی گرداند. شی حاصل به صورت نزولی مرتب شده و اغلب اولین عنصر اتفاق افتاده و به طور پیش فرض مقدار NA را حذف می کند.

```
print(cancer.target_names)
print(df['target'].value_counts())
```

```
['malignant' 'benign']
1.0    357
0.0    212
Name: target, dtype: int64
```

1.8 + 1.7

```
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, stratify=cancer.target, random_state=42)
print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)
```

```
(426, 30)
(143, 30)
(426,)
(143,)
```

1.9

```
classifier = KNeighborsClassifier(n_neighbors = 6)
classifier.fit(X_train, y_train)

print('Accuracy training set: {:.3f}'.format(classifier.score(X_train, y_train)))
print('Accuracy test set: {:.3f}'.format(classifier.score(X_test, y_test)))
```

```
Accuracy training set: 0.948
Accuracy test set: 0.923
```

1.10

```
print("Predictions: {}".format(classifier.predict(X_test)))
```

```
Predictions: [1 0 1 0 1 0 0 0 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
 1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 1 0 1 0 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1 1 1 1 1
 1 0 0 0 0 1 1 1 1 1 0 1 0 1 1 1 1 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 1 1 0 1 0
 1 1 0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1]
```

1.11

در برآوردهای تحت نظارت، تابع model.predict() با توجه به یک مدل آموزش دیده، برچسب یک مجموعه جدید داده را پیش بینی میکند. این روش یک استدلال، داده جدید را می پذیرد، داده های جدید X_new(model.predict(X_new)) و برچسب یاد شده را برای هر شی در آرایه باز می گرداند.

توضیحات کامل تر همراه با یک مثال کاربردی در لینک زیر مورد بررسی قرار گرفت:

https://scipy-lectures.org/packages/scikit-learn/index.html

1.14 + 1.13 + 1.12

```
from sklearn.preprocessing import MinMaxScaler

mxScalar = MinMaxScaler()
X_train = mxScalar.fit_transform(X_train)
X_test = mxScalar.transform(X_test)

classifier.fit(X_train, y_train)

print('MinMaxScaler Accuracy training set: {:.3f}'.format(classifier.score(X_train, y_train)))
print('MinMaxScaler Accuracy test set: {:.3f}'.format(classifier.score(X_test, y_test)))
```

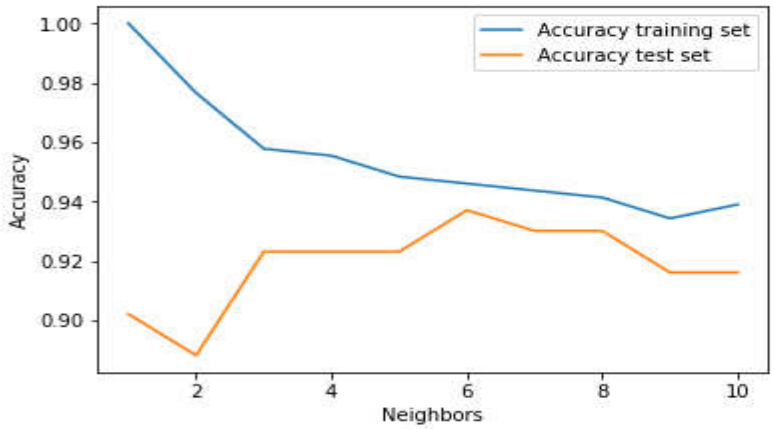
```
MinMaxScaler Accuracy training set: 0.977
MinMaxScaler Accuracy test set: 0.972
```

1.15

```
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, stratify=cancer.target, random_state=66)

training_accuracy = []
test_accuracy = []

neighbors_settings = range(1,11)

for n_neighbors in neighbors_settings:
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    training_accuracy.append(clf.score(X_train, y_train))
    test_accuracy.append(clf.score(X_test, y_test))
    y_pred = clf.predict(X_test)
    print("Accuracy is ", accuracy_score(y_test,y_pred)*100,"% for K:",n_neighbors)

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

```
Accuracy is  90.20979020979021 % for K: 1
Accuracy is  88.81118881118881 % for K: 2
Accuracy is  92.3076923076923 % for K: 3
Accuracy is  92.3076923076923 % for K: 4
Accuracy is  92.3076923076923 % for K: 5
Accuracy is  93.7062937062937 % for K: 6
Accuracy is  93.00699300699301 % for K: 7
Accuracy is  93.00699300699301 % for K: 8
Accuracy is  91.6083916083916 % for K: 9
Accuracy is  91.6083916083916 % for K: 10
```

1.16



1.17

به نظر میرسد با افزایش داده دقت مدل هر بار به سمت بهبود و به داده اصلی نزدیکتر می شود.

سوال 2:

درخت های تصمیم در داده‌کاوی و طبقه‌بندی جایگاهِ ویژه‌ای دارند و بسیاری از الگوریتم‌های طبقه‌بندی بر پایه‌ی این درخت‌ها ساخته شده‌اند. به آن‌ها درخت‌های تصمیم میگویند زیرا می‌توانند یک تصمیمِ خاص(مثلا اینکه به یک شخص وام بدهیم یا نه) را بر اساسِ اطلاعاتِ گذشته اتخاذ کنند.

2.2 + 2.1

```
df = pd.read_csv('dataset_54_vehicle.csv', sep=',')
df.head()
```

TER_RATIO	ELONGATEDNESS	PR.AXIS_RECTANGULARITY	MAX.LENGTH_RECTANGULARITY	SCALED_VARIANCE_MAJOR	SCALED_VARIANCE_MINOR	SCALED
162	42	20	159	176	379	
149	45	19	143	170	330	
207	32	23	158	223	635	
144	46	19	143	160	309	
149	45	19	144	241	325	

2.3

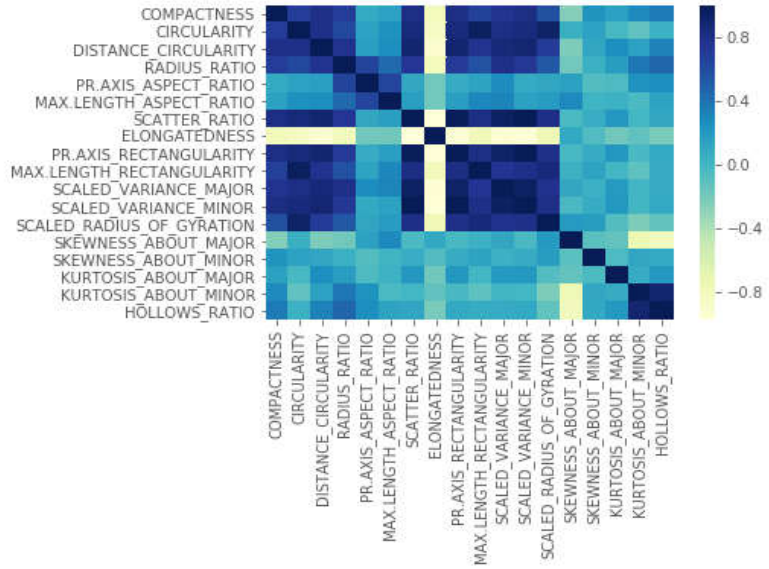
```
print(df['Class'].unique())
```

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

2.4

```
sns.heatmap(df.corr(), cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff261c0d780>
```



2.5

```
x = df.iloc[:,:-1]
print(x.shape)
y = df.iloc[:,-1]
print(y.shape)
```

```
(846, 18)
(846,)
```

2.6

```
X_train, X_test, y_train, y_test = train_test_split(x, y,test_size = 0.2, random_state=42)
```

2.7

```
clf_gini = DecisionTreeClassifier(max_depth = 5, max_features=4, criterion='entropy')
clf_gini.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
            max_features=4, 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')
```

2.10 + 2.9 + 2.8

```
param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 9),
              "min_samples_leaf": randint(1, 9)}

tree = DecisionTreeClassifier()

tree_cv = RandomizedSearchCV(tree, param_dist, cv=5)

tree_cv.fit(x, y)

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': 3, 'min_samples_leaf': 5}
Best score is 0.6867612293144209
```

2.12 + 2.11

```
clf_entropy = DecisionTreeClassifier(criterion = "entropy", max_depth = None, max_features=6, min_samples_leaf=8)
clf_entropy.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=6, max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=8,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

```
y_pred = clf_entropy.predict(X_test)
y_pred
```

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

```
print("Accuracy is ", accuracy_score(y_test,y_pred)*100)
```

```
Accuracy is  72.35294117647058
```

2.13

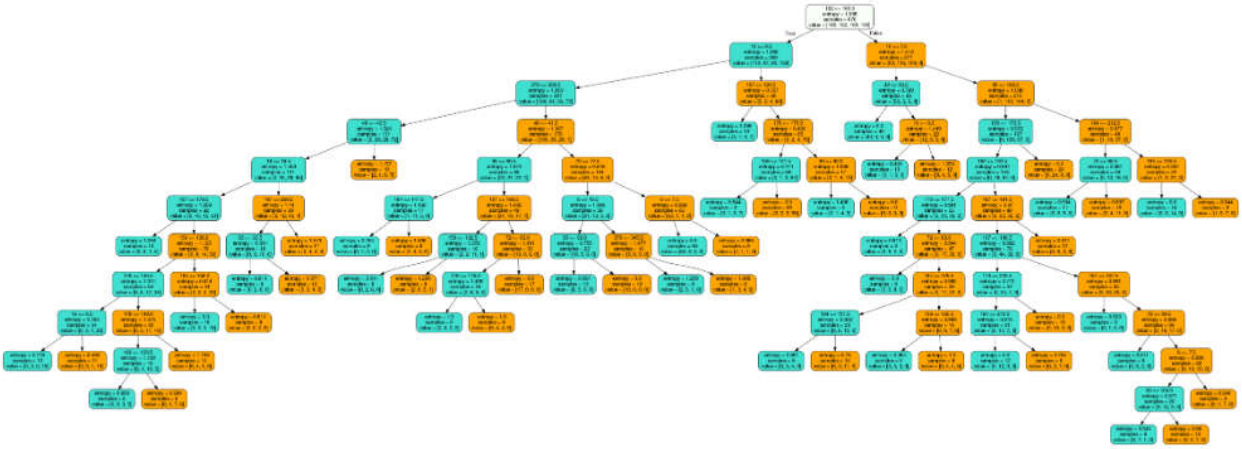
یکی از ویژگی ها در Gradient Boosting است. مزیت استفاده از تقویت کننده این است که پس از افزایش درختان ساخته شده بازیابی نمرات اهمیت برای هر ویژگی نسبتا آسان بوده و در میان تمام درخت های تصمیم گیری درون مدل، میانگین می شود

```
fimportant = dict(zip(df.columns, clf_entropy.feature_importances_))
for key,val in fimportant.items():
    print(key, "=>", val)
```

```
COMPACTNESS => 0.035223190702664366
CIRCULARITY => 0.04378612742658622
DISTANCE_CIRCULARITY => 0.012594675012691752
RADIUS_RATIO => 0.022945474113781967
PR.AXIS_ASPECT_RATIO => 0.03219855324923826
MAX.LENGTH_ASPECT_RATIO => 0.2577608931959796
SCATTER_RATIO => 0.1957302414597378
ELONGATEDNESS => 0.018259364681371425
PR.AXIS_RECTANGULARITY => 0.0
MAX.LENGTH_RECTANGULARITY => 0.03635301401037879
SCALED_VARIANCE_MAJOR => 0.01529097808908382
SCALED_VARIANCE_MINOR => 0.16321269996375318
SCALED_RADIUS_OF_GYRATION => 0.03478185320522066
SKEWNESS_ABOUT_MAJOR => 0.024337661652431668
SKEWNESS_ABOUT_MINOR => 0.02410977811631467
KURTOSIS_ABOUT_MAJOR => 0.023640535372890476
KURTOSIS_ABOUT_MINOR => 0.02839496966409105
HOLLOWS_RATIO => 0.03137999008378426
```

2.16 + 2.15 + 2.14

```
from sklearn import tree

dot_data = tree.export_graphviz(clf_entropy,
                                feature_names=df.iloc[0,0:-1],
                                out_file=None,
                                filled=True,
                                rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)

colors = ('turquoise', 'orange')
edges = collections.defaultdict(list)

for edge in graph.get_edge_list():
    edges[edge.get_source()].append(int(edge.get_destination()))

for edge in edges:
    edges[edge].sort()
    for i in range(2):
        dest = graph.get_node(str(edges[edge][i]))[0]
        dest.set_fillcolor(colors[i])

graph.write_png('tree.png')
Image('tree.png')
```



سوال 3:

3.2 + 3.1

In [1]:
```
from IPython.display import Image

import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.cluster import KMeans

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
matplotlib.style.use('ggplot')
```

In [2]:
```
iris = datasets.load_iris()
iris.data.shape
iris.target.shape
```

Out[2]:
```
(150,)
```

In [3]:
```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target']=iris.target

print(iris.target_names)
df['species'] = df['target'].map({0:iris.target_names[0],1:iris.target_names[1],2:iris.target_names[2]})
df.head()
```

```
['setosa' 'versicolor' 'virginica']
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

```
model = KMeans(n_clusters=3)
model.fit(samples)
```

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

```
labels = model.predict(samples)
labels
```

```
array([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, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)
```

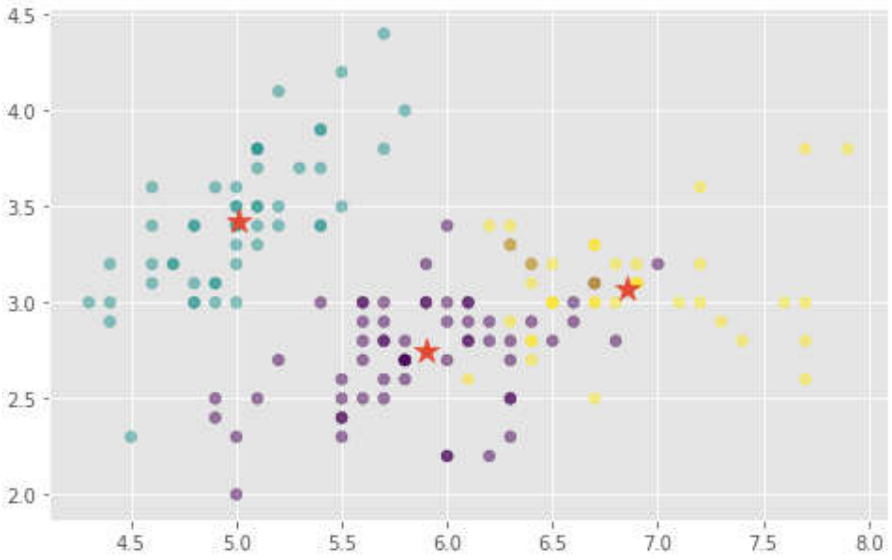
3.3

```
centroids = model.cluster_centers_
centroids
```

```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

3.4

```
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()
```



3.5

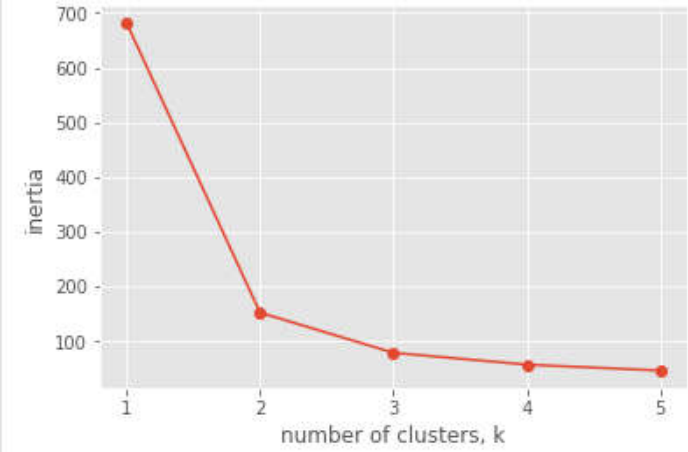
```
print(model.inertia_)
```

```
78.85144142614601
```

3.7 + 3.6

```
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()
```

```
[681.3706, 152.34795176035792, 78.85144142614601, 57.255523809523815, 46.44618205128205]
```



سوال 4:

خوشه بندی سلسله مراتبی، همچنین به عنوان تجزیه و تحلیل خوشه سلسله مراتبی شناخته می شود، یک الگوریتمی است که گروه های مشابه را به گروه های به نام خوشه ها طبقه بندی می کند. نقطه پایانی مجموعه ای از خوشه ها است، جایی که هر خوشه از خوشه های دیگر متمایز است و اشیاء درون هر خوشه به طور گسترده ای مشابه یکدیگرند.

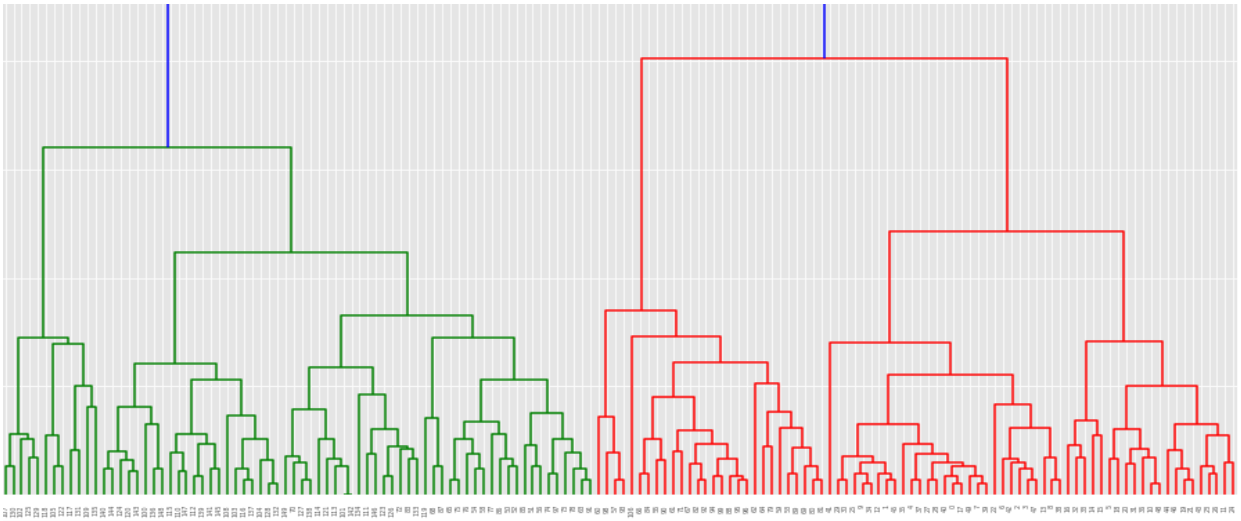
4.1 + 4.2

```
from IPython.display import Image

import pandas as pd
import numpy as np

from sklearn import datasets

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')
```

```
iris = datasets.load_iris()
iris.data.shape
iris.target.shape
```

```
(150,)
```

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

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



4.3

```
clusters = fcluster(linkage_matrix, 6, criterion="distance")
clusters
```

```
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, 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)
```

```
clusters = fcluster(linkage_matrix, 2, criterion='maxclust')
clusters
```

```
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, 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)
```

4.4

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

سوال 5:

در مدل‌های آماری، تحلیل رگرسیون یک فرایند آماری برای تخمین روابط بین متغیرها می‌باشد. این روش شامل تکنیک‌های زیادی برای مدل‌سازی و تحلیل متغیرهای خاص و منحصر بفرد، با تمرکز بر رابطه بین متغیر وابسته و یک یا چند متغیر مستقل، می‌باشد. تحلیل رگرسیون خصوصاً کمک می‌کند در فهم اینکه چگونه مقدار متغیر وابسته با تغییر هرکدام از متغیرهای مستقل و با ثابت بودن دیگر متغیرهای مستقل تغییر می‌کند.

5.1

```
from sklearn.datasets import load_boston
boston_dataset = load_boston()
```

```
print(boston_dataset.keys())
```

```
['data', 'feature_names', 'DESCR', 'target']
```

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

	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

5.2

```
boston['Price'] = boston_dataset.target
```

```
boston.head()
```

	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

5.3

```
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:', array([22.46681692]))
('slope:', array([[-0.34977589,  0.11642402]]))
```

5.4

```
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)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(354, 2)
(152, 2)
(354, 1)
(152, 1)
```

5.5

```
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)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
y_train_predict = lin_model.predict(X_train)
rmse = (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('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
rmse = (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('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 7.65609383626
R2 score is 0.265809345675


The model performance for testing set
--------------------------------------
RMSE is 8.91859167393
R2 score is 0.163491451224
```

5.6 + 5.7+ 5.8 + 5.9

```
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:', array([34.55384088]))
('slope:', array([[-0.95004935]]))
```

```
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)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(354, 1)
(152, 1)
(354, 1)
(152, 1)
```

```
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)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
# model evaluation for training set
y_train_predict = lin_model.predict(X_train)
rmse = (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('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")

# model evaluation for testing set
y_test_predict = lin_model.predict(X_test)
rmse = (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('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 5.9423982329
R2 score is 0.557699059945


The model performance for testing set
--------------------------------------
RMSE is 6.7772343363
R2 score is 0.51696029876
```

می توان نتیجه گرفت که مورد دوم کارایی بالاتری داشت . با توجه به پارامترهای RMSE و همچنین R2

سوال 6:

6.1

```
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, 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,
                1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
                0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
                1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
                0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
                1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
                1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
```

6.2

```
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 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]
```

6.5 + 6.4 + 6.3

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
Confiuse=confusion_matrix(y_test, y_pred)
print (Confiuse)
```

```
array([[44,  3],
       [ 3, 64]], dtype=int64)
```

```
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
```

ماتریس در هم ریختگی به ماتریسی گفته می‌شود که در آن عملکرد الگوریتم‌های مربوطه را نشان می‌دهند. معمولاً چنین نمایشی برای الگوریتم‌های یادگیری با ناظر استفاده می‌شود، اگرچه در یادگیری بدون ناظر نیز کاربرد دارد. در اینجا نیز میزان میانگین عملکرد توابع داده شده اندازه گیری شده است.

6.7 + 6.6

```
from sklearn.preprocessing import normalize
Normal=normalize(Confiuse, norm='l1')
print(Normal)
```

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

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

```
             benign  malignant
benign     0.936170   0.063830
malignant  0.044776   0.955224
```

```
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.   ]
```

6.8

سوال 7:

7.5 + 7.4 + 7.3 + 7.2 + 7.1

```
import pandas as pd
import numpy as np

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

	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

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

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

```
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
```

7.7 + 7.6

این دستورات میزان ارتباطات متغیر های مختلف بایکدیگر را میسنجند.

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

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

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
InvoiceNo												
536370	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
536974	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
537463	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

7.8

```
basket=basket.applymap(lambda x: 1 if x > 0 else 0)
basket.head()
```

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
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

7.10 + 7.9

```
basket=basket.drop("POSTAGE",axis=1)
```

```
frequent_itemsets = apriori(basket, min_support=0.07, use_colnames=True)
frequent_itemsets.head()
```

	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)

7.11

```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.head()
```

	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

7.12

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

	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

شاد باشید.