Perform Principal component analysis and perform clustering using first
3 principal component scores (both heirarchial and k mean clustering(scree plot or elbow curve) and obtain optimum number of clusters and check whether we have obtained same number of clusters with the original data (class column we have ignored at the begining who shows it has 3 clusters)df

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as mp
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import plotly.offline as py
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
```

In [2]:
```
wine_data = pd.read_csv("wine.csv")
wine_data.head(5)
```

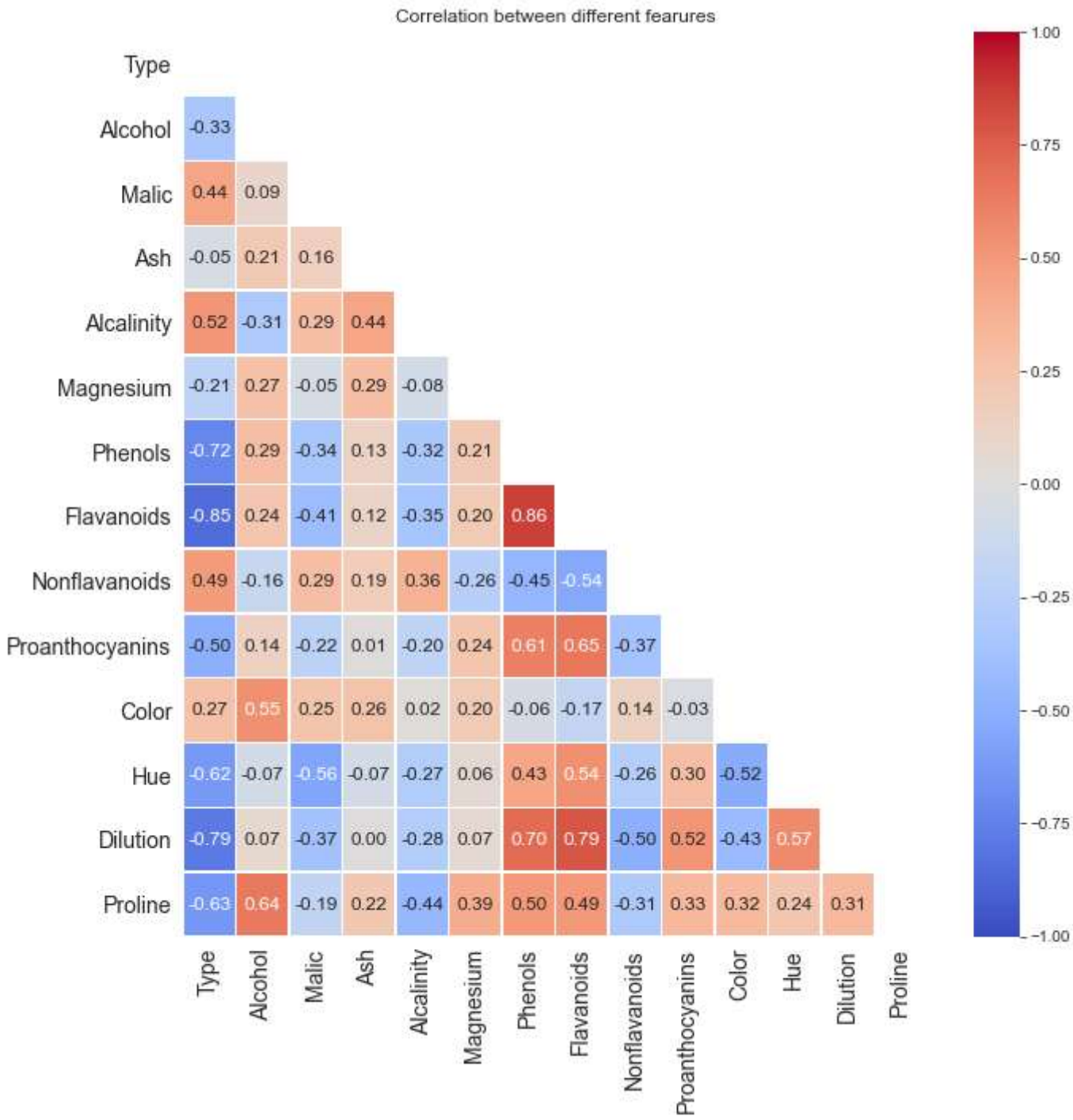
Out[2]:

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Pr
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	

In [3]:
```
wine_data.isnull().any().any()
```

Out[3]: False

In [4]:
```
sns.set_style("white")
matrix = np.triu(wine_data.corr(method="pearson"))
f,ax=plt.subplots(figsize = (matrix.shape[0]*0.75,
                             matrix.shape[1]*0.75))
sns.heatmap(wine_data.corr(method="pearson"),
                annot= True,
                fmt = ".2f",
                ax=ax,
                vmin = -1,
                vmax = 1,
                mask = matrix,
                cmap = "coolwarm",
                linewidth = 0.4,
                linecolor = "white",
                annot_kws={"size": 12})
plt.xticks(rotation=90,size=14)
plt.yticks(rotation=0,size=14)
plt.title('Correlation between different fearures')
plt.show()
```



In [5]:
```
X=wine_data.drop(["Type"],axis=1).values
y=wine_data.Type
```

In [6]:
```
np.shape(X)
```

Out[6]: (178, 13)

In [7]:
```
np.shape(y)
```

Out[7]: (178,)

In [8]:
```
# Data Stardardization
from sklearn.preprocessing import StandardScaler
X_ = StandardScaler().fit_transform(X)
```

In [9]:
```
X_.shape
```

Out[9]: (178, 13)

Computing Rgenvectors and Eigenvalues

In [10]:
```
# Covariance Matrix
mean_vec = np.mean(X_, axis = 0)
cov_mat = (X_ - mean_vec).T.dot((X_ - mean_vec)) / (X_.shape[0]-1)
cov_mat
```

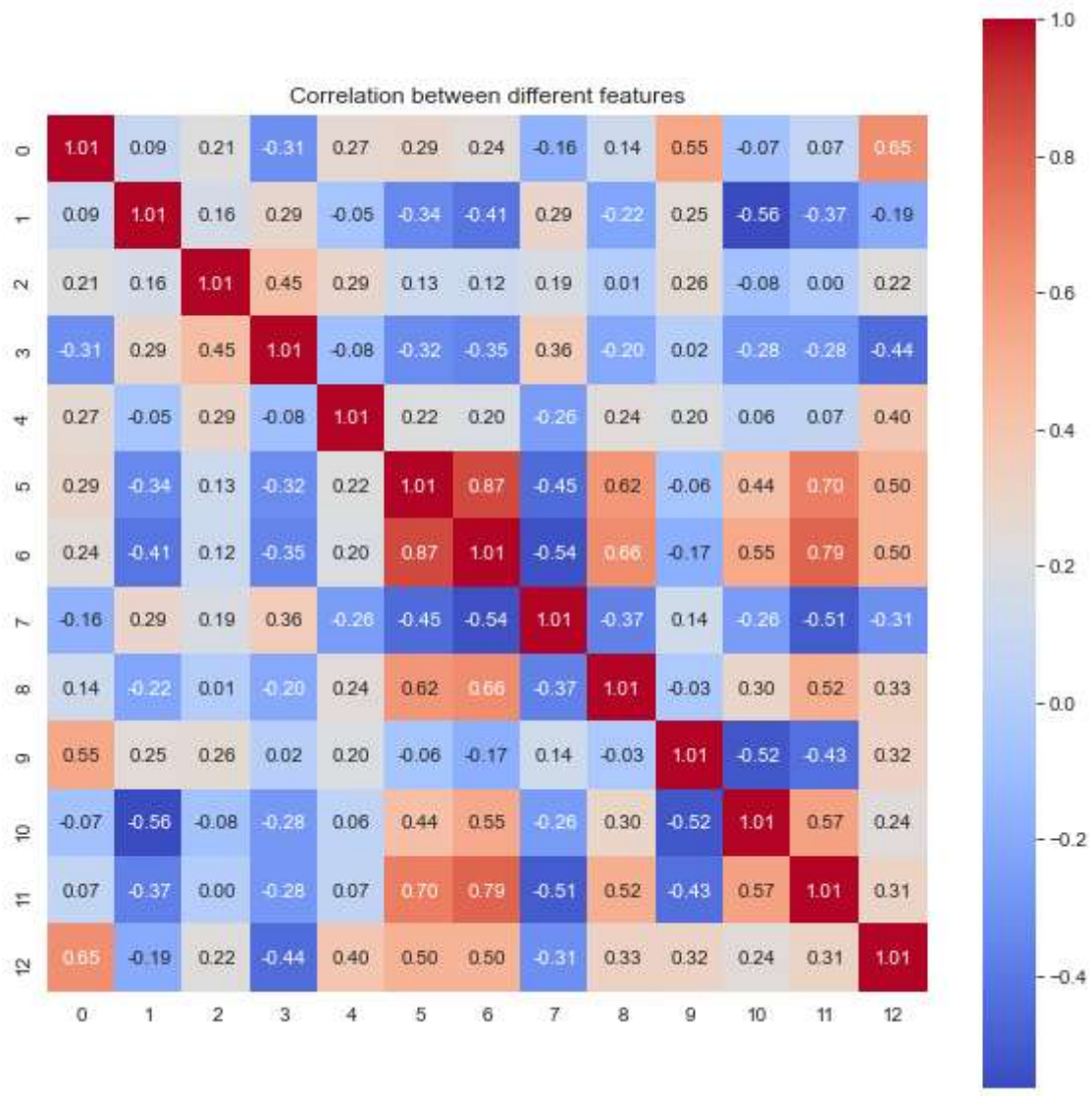
Out[10]:
```
array([[ 1.00564972,  0.09493026,  0.21273976, -0.31198788,  0.27232816,
         0.29073446,  0.23815287, -0.15681042,  0.13747022,  0.549451  ,
        -0.07215255,  0.07275191,  0.64735687],
       [ 0.09493026,  1.00564972,  0.16497228,  0.29013035, -0.05488343,
        -0.3370606 , -0.41332866,  0.29463237, -0.22199334,  0.25039204,
        -0.56446685, -0.37079354, -0.19309537],
       [ 0.21273976,  0.16497228,  1.00564972,  0.44587209,  0.28820583,
         0.12970824,  0.11572743,  0.1872826 ,  0.00970647,  0.2603499 ,
        -0.07508874,  0.00393333,  0.22488969],
       [-0.31198788,  0.29013035,  0.44587209,  1.00564972, -0.0838039 ,
        -0.32292752, -0.353355  ,  0.36396647, -0.19844168,  0.01883781,
        -0.27550299, -0.27833221, -0.44308618],
       [ 0.27232816, -0.05488343,  0.28820583, -0.0838039 ,  1.00564972,
         0.21561254,  0.19688989, -0.25774204,  0.23777643,  0.20107967,
         0.05571118,  0.06637684,  0.39557317],
       [ 0.29073446, -0.3370606 ,  0.12970824, -0.32292752,  0.21561254,
         1.00564972,  0.86944804, -0.45247731,  0.61587304, -0.05544792,
         0.43613151,  0.70390388,  0.50092909],
       [ 0.23815287, -0.41332866,  0.11572743, -0.353355  ,  0.19688989,
         0.86944804,  1.00564972, -0.54093859,  0.65637929, -0.17335329,
         0.54654907,  0.79164133,  0.49698518],
       [-0.15681042,  0.29463237,  0.1872826 ,  0.36396647, -0.25774204,
        -0.45247731, -0.54093859,  1.00564972, -0.36791202,  0.13984265,
        -0.26412347, -0.50611293, -0.31314443],
       [ 0.13747022, -0.22199334,  0.00970647, -0.19844168,  0.23777643,
         0.61587304,  0.65637929, -0.36791202,  1.00564972, -0.02539259,
         0.29721399,  0.52199968,  0.33228346],
       [ 0.549451  ,  0.25039204,  0.2603499 ,  0.01883781,  0.20107967,
        -0.05544792, -0.17335329,  0.13984265, -0.02539259,  1.00564972,
        -0.52476129, -0.43123763,  0.31788599],
       [-0.07215255, -0.56446685, -0.07508874, -0.27550299,  0.05571118,
         0.43613151,  0.54654907, -0.26412347,  0.29721399, -0.52476129,
         1.00564972,  0.56866303,  0.23751782],
       [ 0.07275191, -0.37079354,  0.00393333, -0.27833221,  0.06637684,
         0.70390388,  0.79164133, -0.50611293,  0.52199968, -0.43123763,
         0.56866303,  1.00564972,  0.31452809],
       [ 0.64735687, -0.19309537,  0.22488969, -0.44308618,  0.39557317,
         0.50092909,  0.49698518, -0.31314443,  0.33228346,  0.31788599,
         0.23751782,  0.31452809,  1.00564972]])
```

In [11]:
```
# NumPy covariance matrix
print('NumPy covariance matrix: \n%s' %np.cov(X_.T))
```

```
NumPy covariance matrix:
[[ 1.00564972  0.09493026  0.21273976 -0.31198788  0.27232816  0.29073446
   0.23815287 -0.15681042  0.13747022  0.549451   -0.07215255  0.07275191
   0.64735687]
 [ 0.09493026  1.00564972  0.16497228  0.29013035 -0.05488343 -0.3370606
  -0.41332866  0.29463237 -0.22199334  0.25039204 -0.56446685 -0.37079354
  -0.19309537]
 [ 0.21273976  0.16497228  1.00564972  0.44587209  0.28820583  0.12970824
   0.11572743  0.1872826   0.00970647  0.2603499  -0.07508874  0.00393333
   0.22488969]
 [-0.31198788  0.29013035  0.44587209  1.00564972 -0.0838039  -0.32292752
  -0.353355    0.36396647 -0.19844168  0.01883781 -0.27550299 -0.27833221
  -0.44308618]
 [ 0.27232816 -0.05488343  0.28820583 -0.0838039   1.00564972  0.21561254
   0.19688989 -0.25774204  0.23777643  0.20107967  0.05571118  0.06637684
   0.39557317]
 [ 0.29073446 -0.3370606   0.12970824 -0.32292752  0.21561254  1.00564972
   0.86944804 -0.45247731  0.61587304 -0.05544792  0.43613151  0.70390388
   0.50092909]
 [ 0.23815287 -0.41332866  0.11572743 -0.353355    0.19688989  0.86944804
   1.00564972 -0.54093859  0.65637929 -0.17335329  0.54654907  0.79164133
   0.49698518]
 [-0.15681042  0.29463237  0.1872826   0.36396647 -0.25774204 -0.45247731
  -0.54093859  1.00564972 -0.36791202  0.13984265 -0.26412347 -0.50611293
  -0.31314443]
 [ 0.13747022 -0.22199334  0.00970647 -0.19844168  0.23777643  0.61587304
   0.65637929 -0.36791202  1.00564972 -0.02539259  0.29721399  0.52199968
   0.33228346]
 [ 0.549451    0.25039204  0.2603499   0.01883781  0.20107967 -0.05544792
  -0.17335329  0.13984265 -0.02539259  1.00564972 -0.52476129 -0.43123763
   0.31788599]
 [-0.07215255 -0.56446685 -0.07508874 -0.27550299  0.05571118  0.43613151
   0.54654907 -0.26412347  0.29721399 -0.52476129  1.00564972  0.56866303
   0.23751782]
 [ 0.07275191 -0.37079354  0.00393333 -0.27833221  0.06637684  0.70390388
   0.79164133 -0.50611293  0.52199968 -0.43123763  0.56866303  1.00564972
   0.31452809]
 [ 0.64735687 -0.19309537  0.22488969 -0.44308618  0.39557317  0.50092909
   0.49698518 -0.31314443  0.33228346  0.31788599  0.23751782  0.31452809
   1.00564972]]
```

In [12]:
```
plt.figure(figsize=(10,10))
sns.heatmap(cov_mat, vmax=1, fmt = '.2f',square=True,annot=True,cmap='coolwarm')

plt.title('Correlation between different features')
```

Out[12]: Text(0.5, 1.0, 'Correlation between different features')



Eigen Decomposition of a covariance matrix

In [13]:
```
eigvals, eigvecs = np.linalg.eig(cov_mat)
print('Eigenvectors \n%s' %eigvecs)
print('\nEigenvalues \n%s' %eigvals)
```

```
Eigenvectors
[[-0.1443294   0.48365155 -0.20738262  0.0178563  -0.26566365  0.21353865
   0.05639636 -0.01496997  0.39613926 -0.26628645 -0.50861912 -0.22591696
   0.21160473]
 [ 0.24518758  0.22493093  0.08901289 -0.53689028  0.03521363  0.53681385
  -0.42052391 -0.02596375  0.06582674  0.12169604  0.07528304  0.07648554
  -0.30907994]
 [ 0.00205106  0.31606881  0.6262239   0.21417556 -0.14302547  0.15447466
   0.14917061  0.14121803 -0.17026002 -0.04962237  0.30769445 -0.49869142
  -0.02712539]
 [ 0.23932041 -0.0105905   0.61208035 -0.06085941  0.06610294 -0.10082451
   0.28696914 -0.09168285  0.42797018 -0.05574287 -0.20044931  0.47931378
   0.05279942]
 [-0.14199204  0.299634    0.13075693  0.35179658  0.72704851  0.03814394
  -0.3228833  -0.05677422 -0.15636143  0.06222011 -0.27140257  0.07128891
   0.06787022]
 [-0.39466085  0.06503951  0.14617896 -0.19806835 -0.14931841 -0.0841223
   0.02792498  0.46390791 -0.40593409 -0.30388245 -0.28603452  0.30434119
  -0.32013135]
 [-0.4229343  -0.00335981  0.1506819  -0.15229479 -0.10902584 -0.01892002
   0.06068521 -0.83225706 -0.18724536 -0.04289883 -0.04957849 -0.02569409
  -0.16315051]
 [ 0.2985331   0.02877949  0.17036816  0.20330102 -0.50070298 -0.25859401
  -0.59544729 -0.11403985 -0.23328465  0.04235219 -0.19550132  0.11689586
   0.21553507]
 [-0.31342949  0.03930172  0.14945431 -0.39905653  0.13685982 -0.53379539
  -0.37213935  0.11691707  0.36822675 -0.09555303  0.20914487 -0.23736257
   0.1341839 ]
 [ 0.0886167   0.52999567 -0.13730621 -0.06592568 -0.07643678 -0.41864414
   0.22771214  0.0119928  -0.03379692  0.60422163 -0.05621752  0.0318388
  -0.29077518]
 [-0.29671456 -0.27923515  0.08522192  0.42777141 -0.17361452  0.10598274
  -0.23207564  0.08988884  0.43662362  0.259214   -0.08582839 -0.04821201
  -0.52239889]
 [-0.37616741 -0.16449619  0.16600459 -0.18412074 -0.10116099  0.26585107
   0.0447637   0.15671813 -0.07810789  0.60095872 -0.1372269   0.0464233
   0.52370587]
 [-0.28675223  0.36490283 -0.12674592  0.23207086 -0.1578688   0.11972557
  -0.0768045  -0.01444734  0.12002267 -0.07940162  0.57578611  0.53926983
   0.162116  ]]

Eigenvalues
[4.73243698 2.51108093 1.45424187 0.92416587 0.85804868 0.64528221
 0.55414147 0.10396199 0.35046627 0.16972374 0.29051203 0.22706428
 0.25232001]
```

Selecting Principal Components

In order to decide which eigenvector(s) can dropped without losing too much information for the construction of lower-dimensional subspace, we need to inspect the corresponding eigenvalues: The eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data; those are the ones can be dropped.

In [14]:
```
eigPairs = [(np.abs(eigvals[i]), eigvecs[:,i]) for i in range(len(eigvals))]
```

In [15]:
```
eigPairs.sort(key = lambda x:x[0], reverse = True)
print('Eigenvalues in descending order:')
for i in eigPairs:
    print(i[0])
```

```
Eigenvalues in descending order:
4.732436977583599
2.511080929645123
1.454241867846469
0.9241658668248737
0.8580486765371119
0.6452822124678533
0.5541414662457839
0.3504662749462539
0.29051203269397713
0.2523200103608252
0.22706428173088533
0.16972373898012164
0.10396199182075357
```

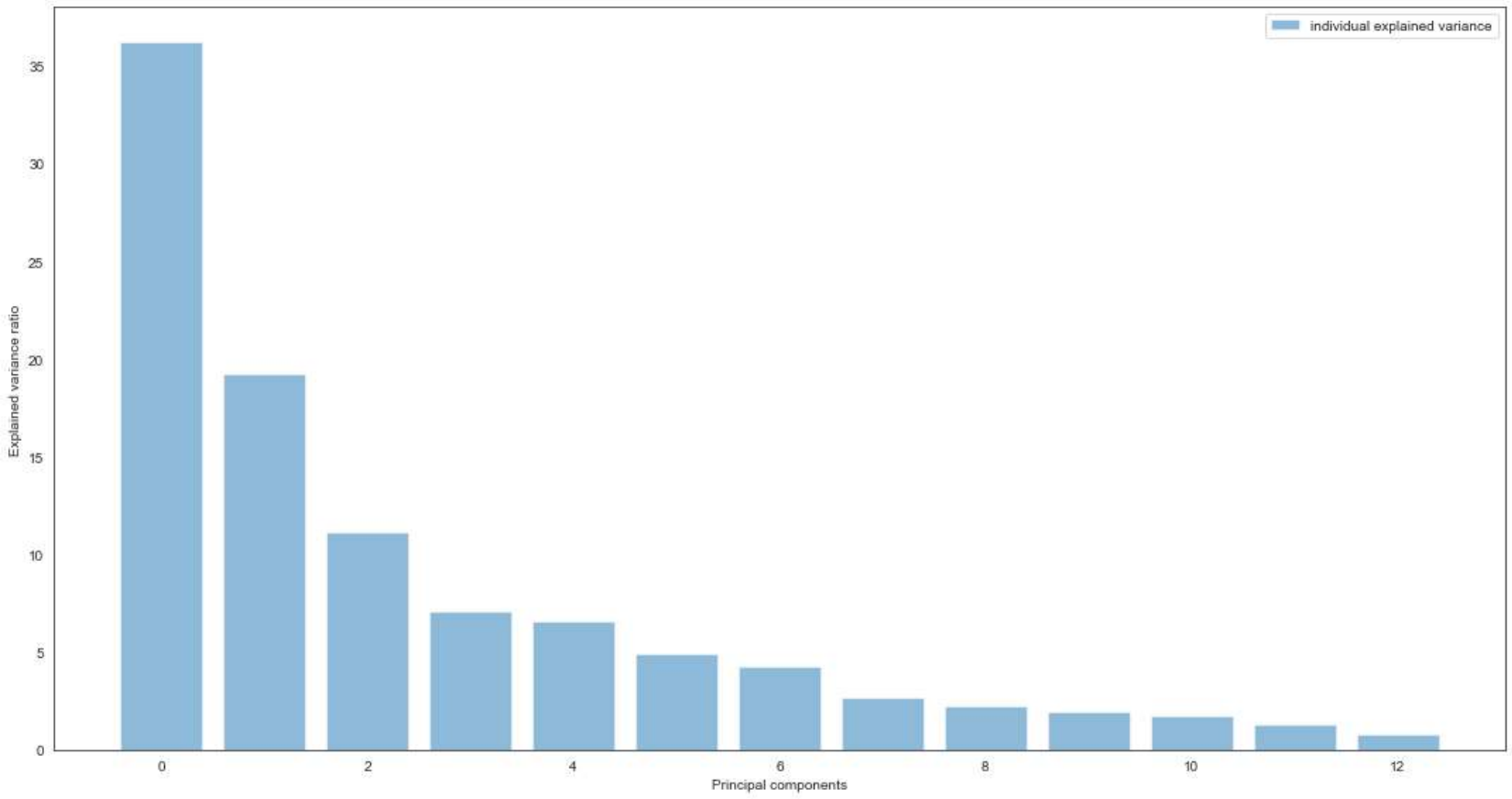
Explained Variance

After sorting the eigenpairs, the next question is "how many principal components are we going to choose for our new feature subspace?" A useful measure is the so-called "explained variance," which can be calculated from the eigenvalues. The explained variance tells us how much information (variance) can be attributed to each of the principal components.

In [16]:
```
exp_var = [(i/sum(eigvals))*100 for i in sorted(eigvals, reverse = True)]
exp_var
```

Out[16]:
```
[36.19884809992636,
 19.207490257008914,
 11.123630536249989,
 7.069030182714021,
 6.563293679648602,
 4.935823319222555,
 4.23867932262332,
 2.680748948378862,
 2.2221534047897102,
 1.9300190939440798,
 1.736835689989916,
 1.2982325756042132,
 0.7952148898994538]
```

In [17]:
```
plt.figure(figsize=(15, 8))

plt.bar(range(13), exp_var, alpha=0.5, align='center',
            label='individual explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
```



The plot above clearly shows that maximum variance (somewhere around 36%) can be explained by the first principal component alone.

Projection Matrix

The construction of the projection matrix that will be used to transform the wine's data into the new feature subspace. Suppose only 1st and 2nd principal component shares the maximum amount of information say around 90%.Hence we can drop other components. Here, we are reducing the 13-dimensional feature space to a 2-dimensional feature subspace, by choosing the “top 2” eigenvectors with the highest eigenvalues to construct our d×k-dimensional eigenvector matrix W

In [18]:
```
matrix_w = np.hstack((eigPairs[0][1].reshape(13,1),
                      eigPairs[1][1].reshape(13,1)
                     ))

matrix_w
```

Out[18]:
```
array([[-0.1443294 ,  0.48365155],
       [ 0.24518758,  0.22493093],
       [ 0.00205106,  0.31606881],
       [ 0.23932041, -0.0105905 ],
       [-0.14199204,  0.299634  ],
       [-0.39466085,  0.06503951],
       [-0.4229343 , -0.00335981],
       [ 0.2985331 ,  0.02877949],
       [-0.31342949,  0.03930172],
       [ 0.0886167 ,  0.52999567],
       [-0.29671456, -0.27923515],
       [-0.37616741, -0.16449619],
       [-0.28675223,  0.36490283]])
```

Projection Onto the New Feature Space

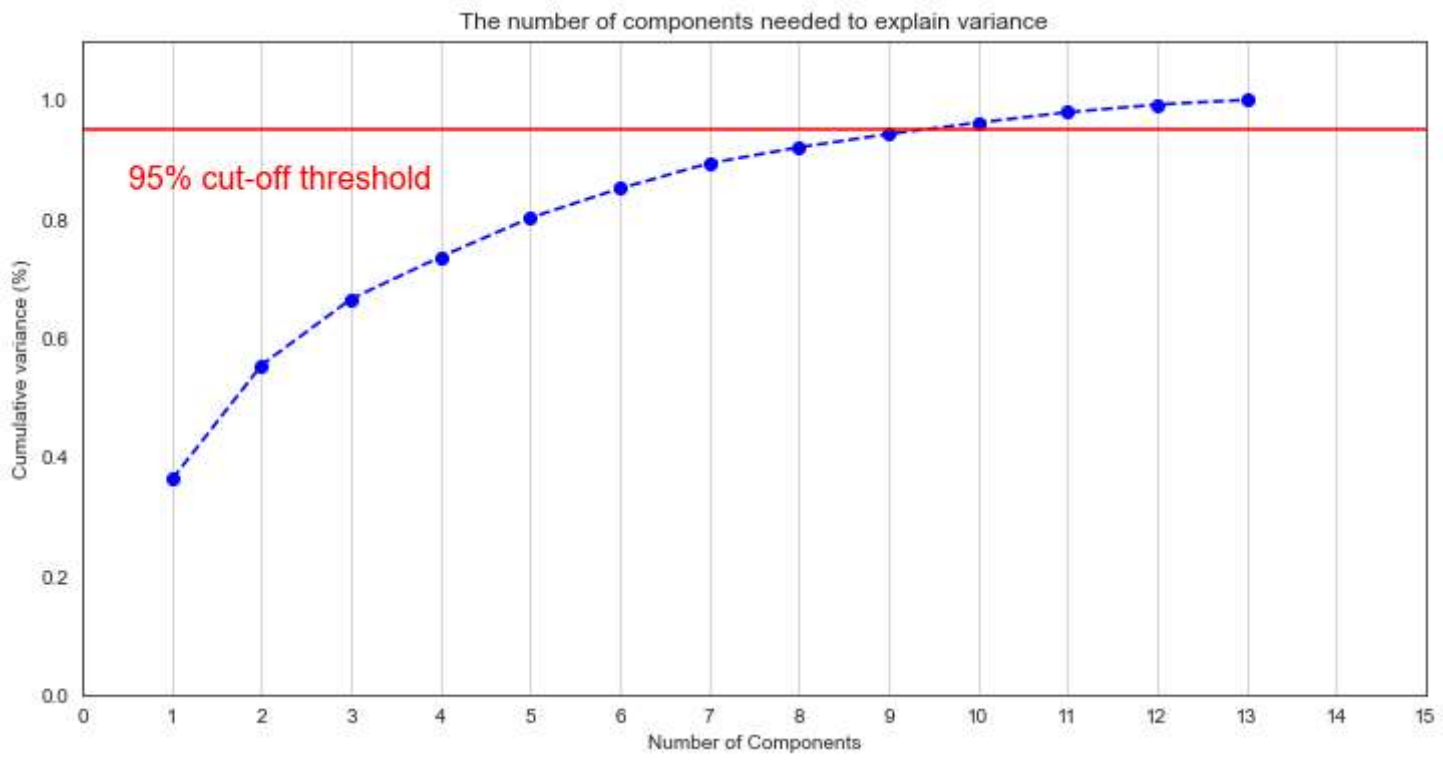
In this last step we will use the 7×2-dimensional projection matrix W to transform our samples onto the new subspace via the equation Y=X×W

In [19]:
```
Y = X_.dot(matrix_w)
Y
```

Out[19]:
```
array([[-3.31675081,  1.44346263],
       [-2.20946492, -0.33339289],
       [-2.51674015,  1.0311513 ],
       [-3.75706561,  2.75637191],
       [-1.00890849,  0.86983082],
       [-3.05025392,  2.12240111],
       [-2.44908967,  1.17485013],
       [-2.05943687,  1.60896307],
       [-2.5108743 ,  0.91807096],
       [-2.75362819,  0.78943767],
       [-3.47973668,  1.30233324],
       [-1.7547529 ,  0.61197723],
       [-2.11346234,  0.67570634],
       [-3.45815682,  1.13062988],
       [-4.31278391,  2.09597558],
       [-2.3051882 ,  1.66255173],
       [-2.17195527,  2.32730534],
       [-1.89897118,  1.63136888],
       [-3.54198508,  2.51834367],
```

Scree plot

In [20]:
```
from sklearn.decomposition import PCA
pca = PCA().fit(X_)

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12,6)

fig, ax = plt.subplots()
xi = np.arange(1, 14, step=1)
y1 = np.cumsum(pca.explained_variance_ratio_)

plt.ylim(0.0,1.1)
plt.plot(xi, y1, marker='o', linestyle='--', color='b')

plt.xlabel('Number of Components')
plt.xticks(np.arange(0, 16, step=1)) #change from 0-based array index to 1-based human-readable label
plt.ylabel('Cumulative variance (%)')
plt.title('The number of components needed to explain variance')

plt.axhline(y=0.95, color='r', linestyle='-')
plt.text(0.5, 0.85, '95% cut-off threshold', color = 'red', fontsize=16)

ax.grid(axis='x')
plt.show()
```



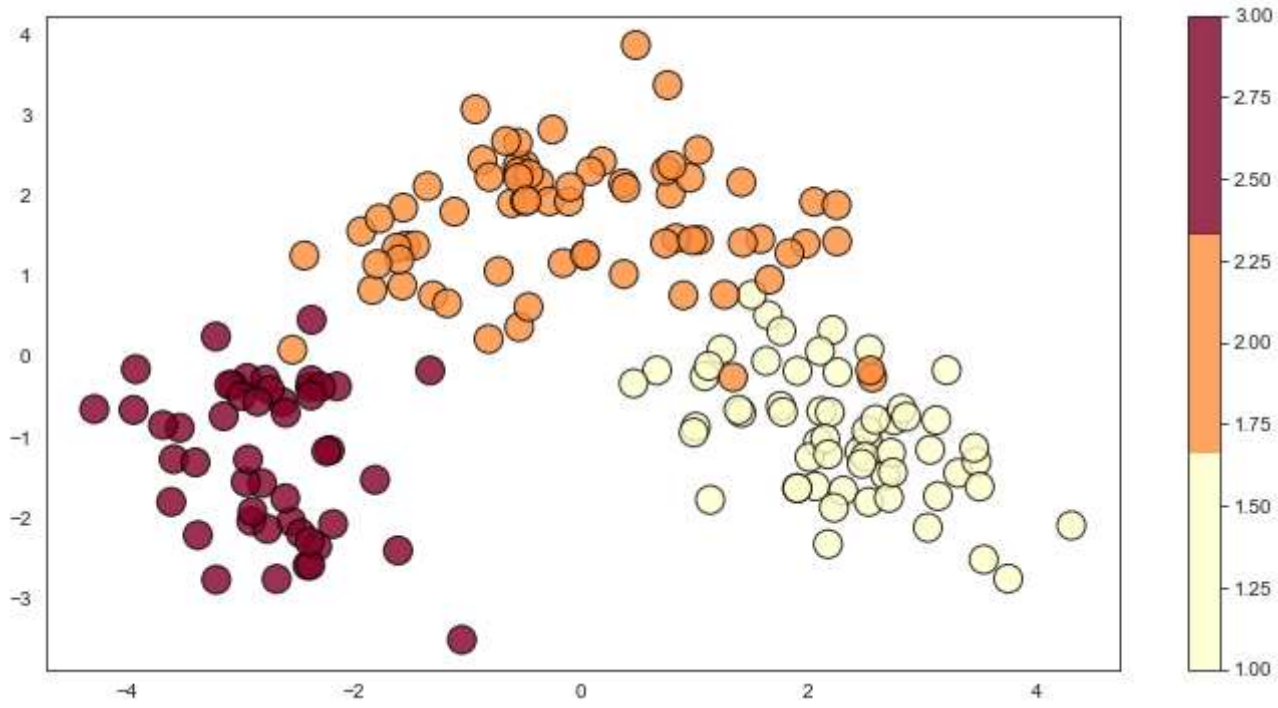
The above plot shows almost 95% variance by the first 9 components. Therfore we can drop last 4 components.

In [21]:
```
PCA_ = PCA(n_components=9)
y_pca = PCA_.fit_transform(X_)
y_pca
```

Out[21]:
```
array([[ 3.31675081e+00, -1.44346263e+00, -1.65739045e-01, ...,
         5.96426546e-01,  6.51390947e-02,  6.41442706e-01],
       [ 2.20946492e+00,  3.33392887e-01, -2.02645737e+00, ...,
         5.37756128e-02,  1.02441595e+00, -3.08846753e-01],
       [ 2.51674015e+00, -1.03115130e+00,  9.82818670e-01, ...,
         4.24205451e-01, -3.44216131e-01, -1.17783447e+00],
       ...,
       [-2.67783946e+00, -2.76089913e+00, -9.40941877e-01, ...,
         6.79235406e-01,  4.70238043e-02,  1.22214687e-03],
       [-2.38701709e+00, -2.29734668e+00, -5.50696197e-01, ...,
         6.33975271e-01,  3.90828774e-01,  5.74476725e-02],
       [-3.20875816e+00, -2.76891957e+00,  1.01391366e+00, ...,
         5.74125710e-03, -2.92913734e-01,  7.41660423e-01]])
```

In [22]:
```
# Creating a scatter plot of the datapoints
plt.scatter(y_pca[:, 0], y_pca[:, 1], c=y, s =200,edgecolor='black', alpha=0.8,
            cmap=plt.cm.get_cmap("YlOrRd",3))
plt.colorbar()
```

Out[22]:
```
<matplotlib.colorbar.Colorbar at 0x255342a2a60>
```

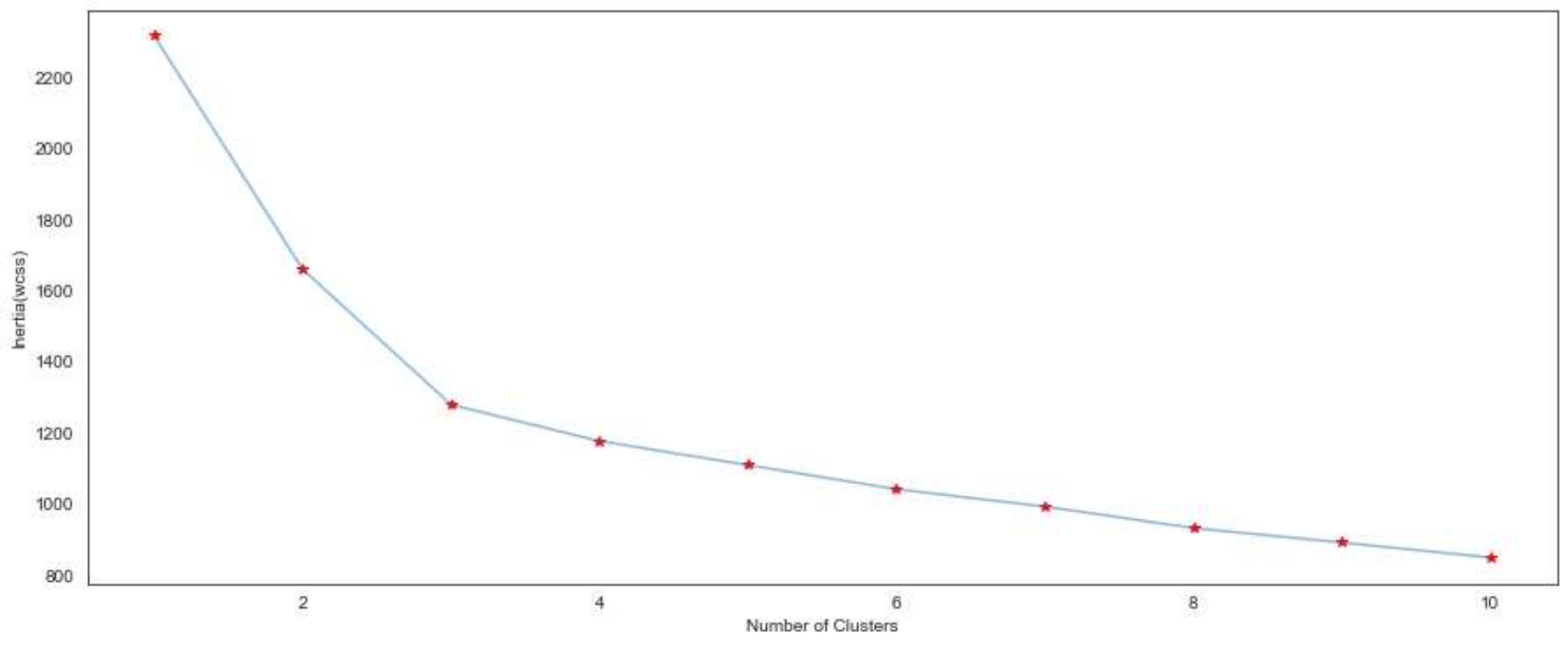


K means clustering using 3 components

In [23]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

wcss =[]
s_scores = []
for i in range(1,11): # Silhoutte score reqy=uires more than one cluster otherwise it will throw an error
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=111, algorithm = 'elkan')
    kmeans.fit(X_)
    #print(np.unique(kmeans.labels_))
    if i!=1:
        silhouette_avg = silhouette_score(X_, kmeans.labels_)
    else:
        silhouette_avg=0

    print("K-Means: for n_clusters =", i, ", the average silhouette_score is", silhouette_avg)

    wcss.append(kmeans.inertia_)
    s_scores.append(silhouette_avg)
```

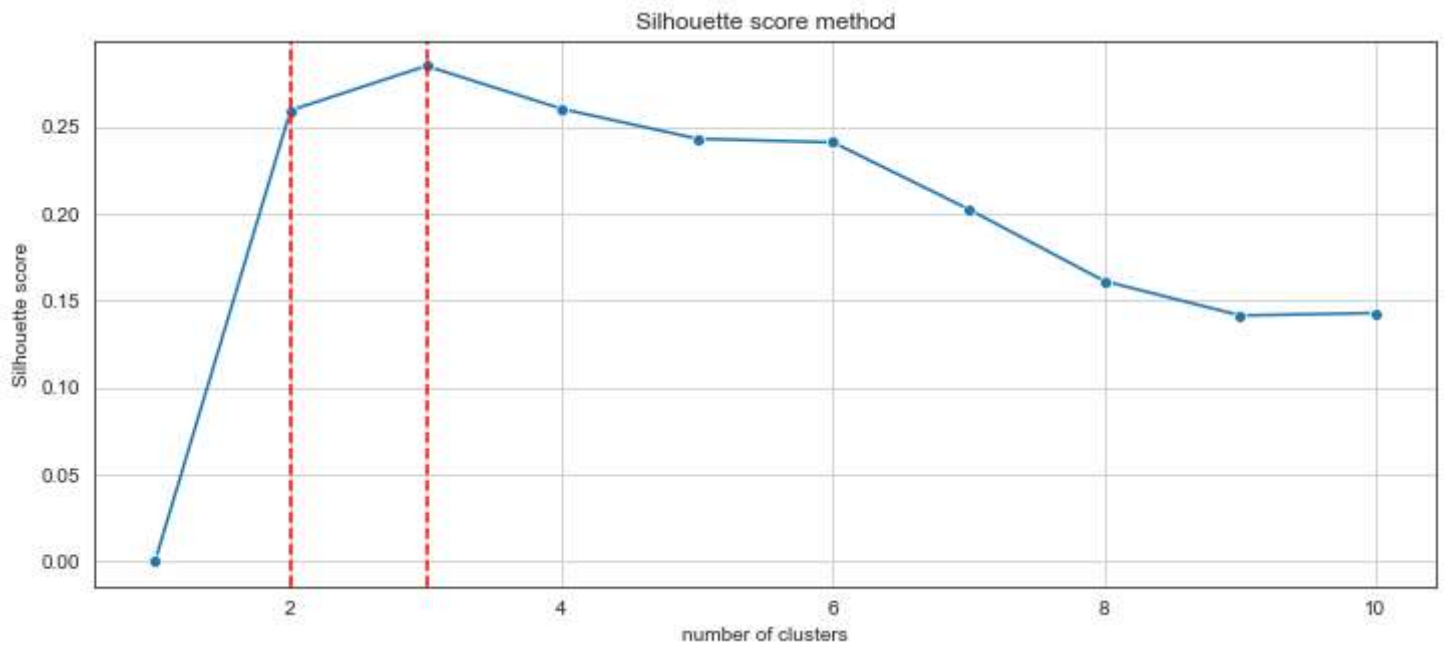
```
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:973: RuntimeWarning:

algorithm='elkan' doesn't make sense for a single cluster. Using 'full' instead.


K-Means: for n_clusters = 1 , the average silhouette_score is 0
K-Means: for n_clusters = 2 , the average silhouette_score is 0.25931695553182543
K-Means: for n_clusters = 3 , the average silhouette_score is 0.2848589191898987
K-Means: for n_clusters = 4 , the average silhouette_score is 0.2601703522370452
K-Means: for n_clusters = 5 , the average silhouette_score is 0.24300081324876488
K-Means: for n_clusters = 6 , the average silhouette_score is 0.24092425943530021
K-Means: for n_clusters = 7 , the average silhouette_score is 0.2024588283961019
K-Means: for n_clusters = 8 , the average silhouette_score is 0.1611820745585241
K-Means: for n_clusters = 9 , the average silhouette_score is 0.14145251149897328
K-Means: for n_clusters = 10 , the average silhouette_score is 0.14285678046616243
```

In [24]:
```
plt.figure(1 , figsize = (15 ,6))
plt.plot(np.arange(1 , 11) , wcss , '*', color ='red')
plt.plot(np.arange(1 , 11) , wcss , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia(wcss)')
plt.show()
```



In [25]:
```
fig, ax = plt.subplots(figsize=(12,5))
ax = sns.lineplot(np.arange(1 , 11), s_scores, marker='o', ax=ax)
ax.set_title("Silhouette score method")
ax.set_xlabel("number of clusters")
ax.set_ylabel("Silhouette score")
ax.axvline(2, ls="--", c="red")
ax.axvline(3, ls="--", c="red")
plt.grid()
plt.show()
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument w
ill be `data`, and passing other arguments without an explicit keyword will result in an error or misinterp
retation.
```



In [26]:
```
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=111, algorithm = 'elkan')
y_kmeans = kmeans.fit_predict(X_)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

K_means_ = wine_data.drop(["Type"],axis=1)
K_means_["Clusters"] = labels
K_means_.head(5)
```

Out[26]:

	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

Principal component analysis

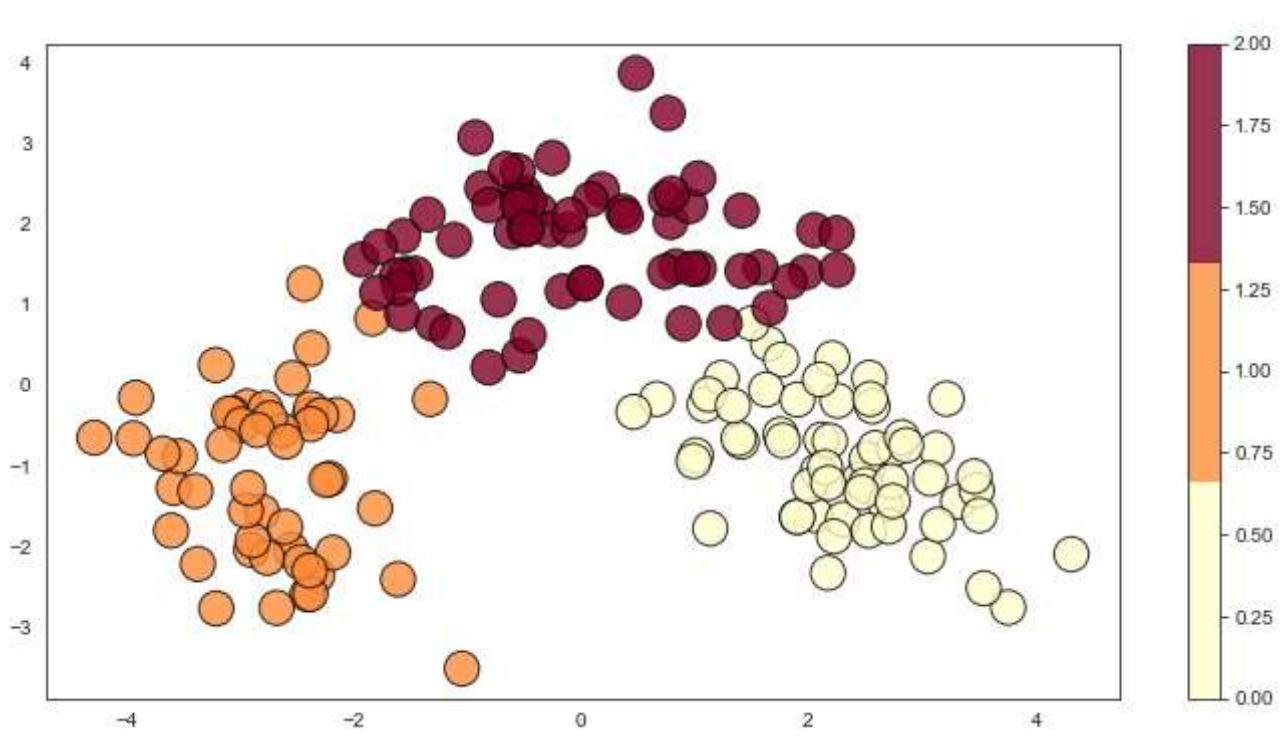
In [28]:
```
PCA_kMeans = PCA(n_components=3)
y_pca_kmeans = PCA_kMeans.fit_transform(X_)
y_pca_kmeans

df_pca_kmeans = pd.DataFrame(data=y_pca_kmeans, columns=['PCA1', 'PCA2', 'PCA3'])
df_pca_kmeans['Clusters'] = labels
df1 = df_pca_kmeans['PCA1'].values
df2 = df_pca_kmeans['PCA2'].values
df3 = df_pca_kmeans['PCA3'].values
colors = df_pca_kmeans["Clusters"]
df_pca_kmeans.tail(10)
```

Out[28]:

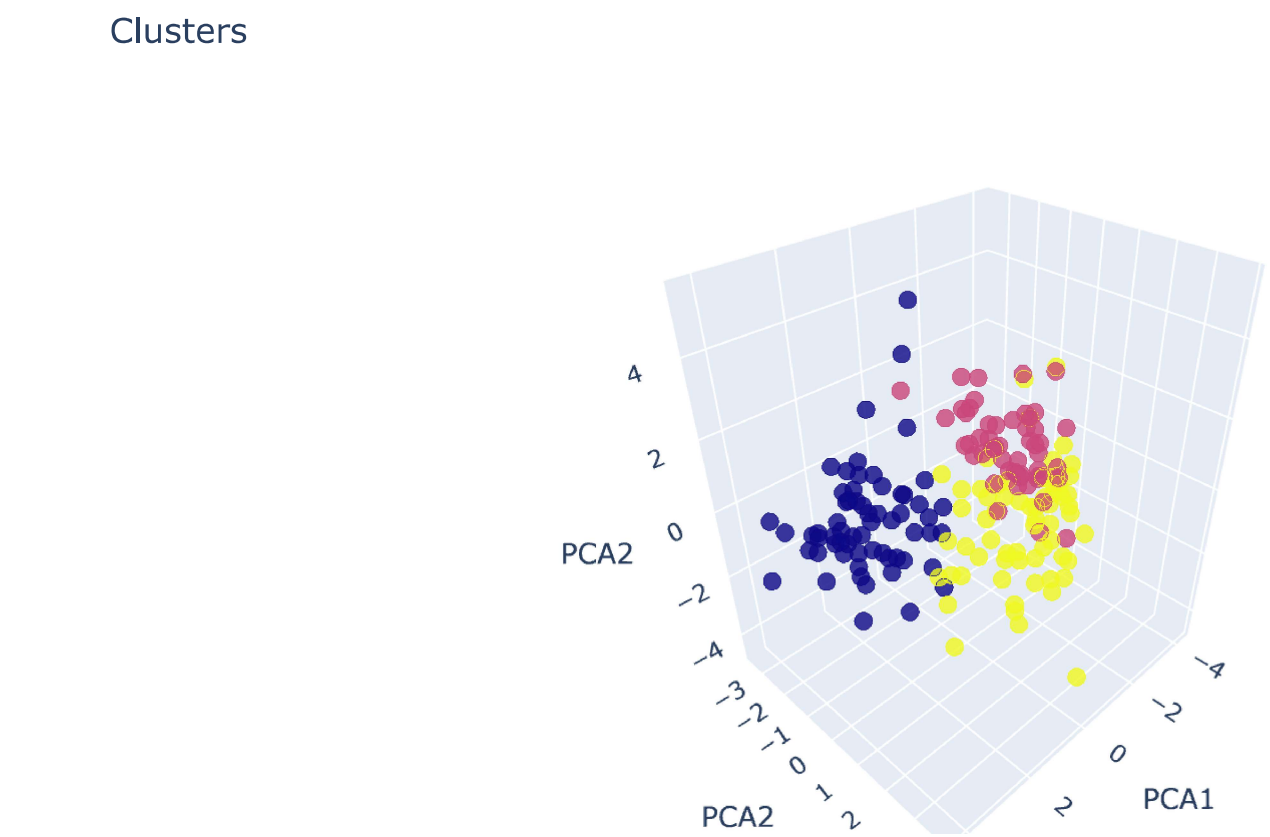
	PCA1	PCA2	PCA3	Clusters
168	-2.181413	-2.077537	0.763783	1
169	-2.380928	-2.588667	1.418044	1
170	-3.211617	0.251249	-0.847129	1
171	-3.677919	-0.847748	-1.339420	1
172	-2.465556	-2.193798	-0.918781	1
173	-3.370524	-2.216289	-0.342570	1
174	-2.601956	-1.757229	0.207581	1
175	-2.677839	-2.760899	-0.940942	1
176	-2.387017	-2.297347	-0.550696	1
177	-3.208758	-2.768920	1.013914	1

In [29]:
```
plt.scatter(df1,df2,c=colors, edgecolor='black', alpha=0.8,  cmap=plt.cm.get_cmap("YlOrRd",3),s=300)
plt.colorbar()
```

Out[29]: <matplotlib.colorbar.Colorbar at 0x25534ca0370>

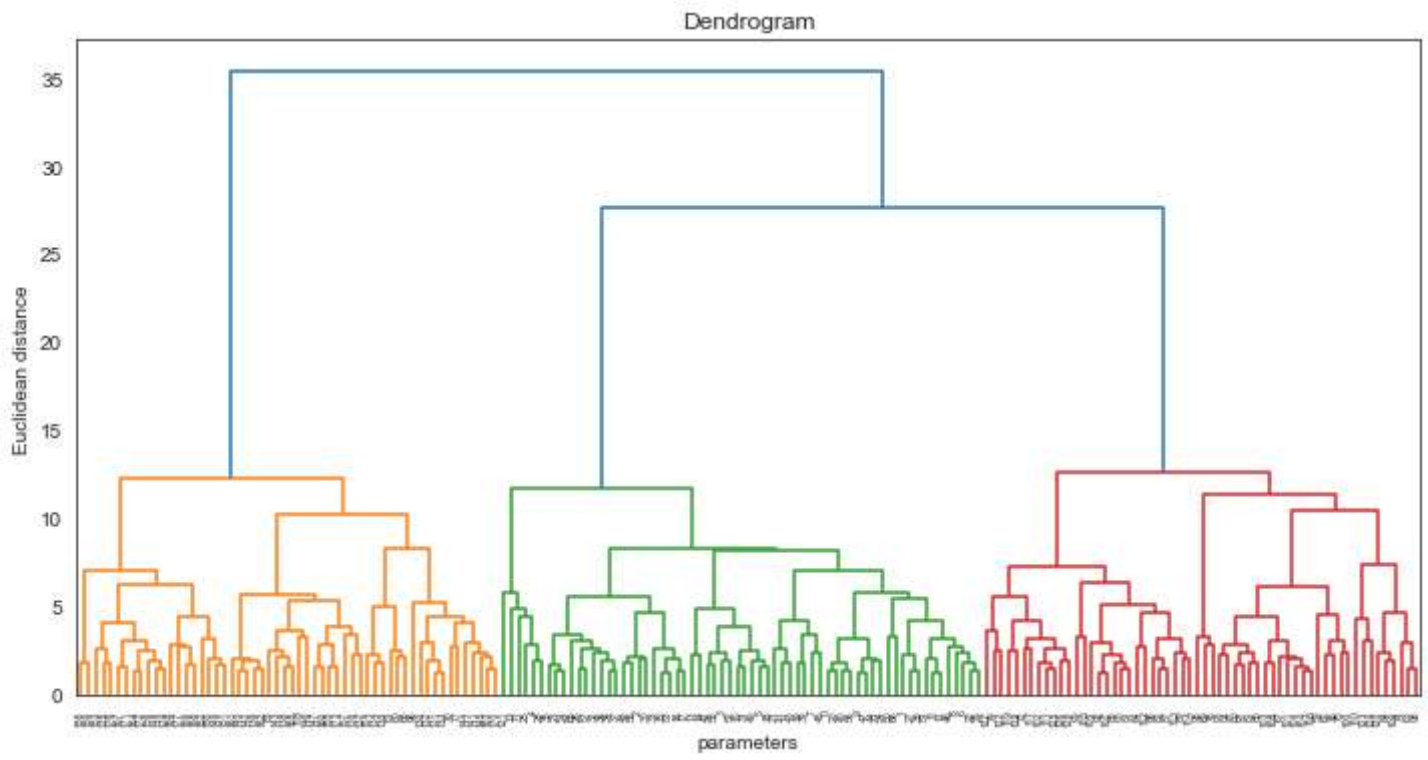


In [30]:
```
trace1 = go.Scatter3d(
    x= df_pca_kmeans['PCA1'],
    y= df_pca_kmeans['PCA2'],
    z= df_pca_kmeans['PCA3'],
   # z1= crime_data['UrbanPop'],
    mode='markers',
     marker=dict(
        color = colors,
        size= 5,
        line=dict(
            color= colors,
            width= 12
        ),
        opacity=0.8
     )
)
data = [trace1]
layout = go.Layout(
    title= 'Clusters',
    scene = dict(
            xaxis = dict(title  = 'PCA1'),
            yaxis = dict(title  = 'PCA2'),
            zaxis = dict(title  = 'PCA2')
        )
)
fig = go.Figure(data=data, layout=layout)
py.offline.iplot(fig)
```



Hierarchical clustering

In [31]:
```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X_, method = 'ward'))

plt.title('Dendrogram')
plt.xlabel('parameters')
plt.ylabel('Euclidean distance')
plt.show()
```



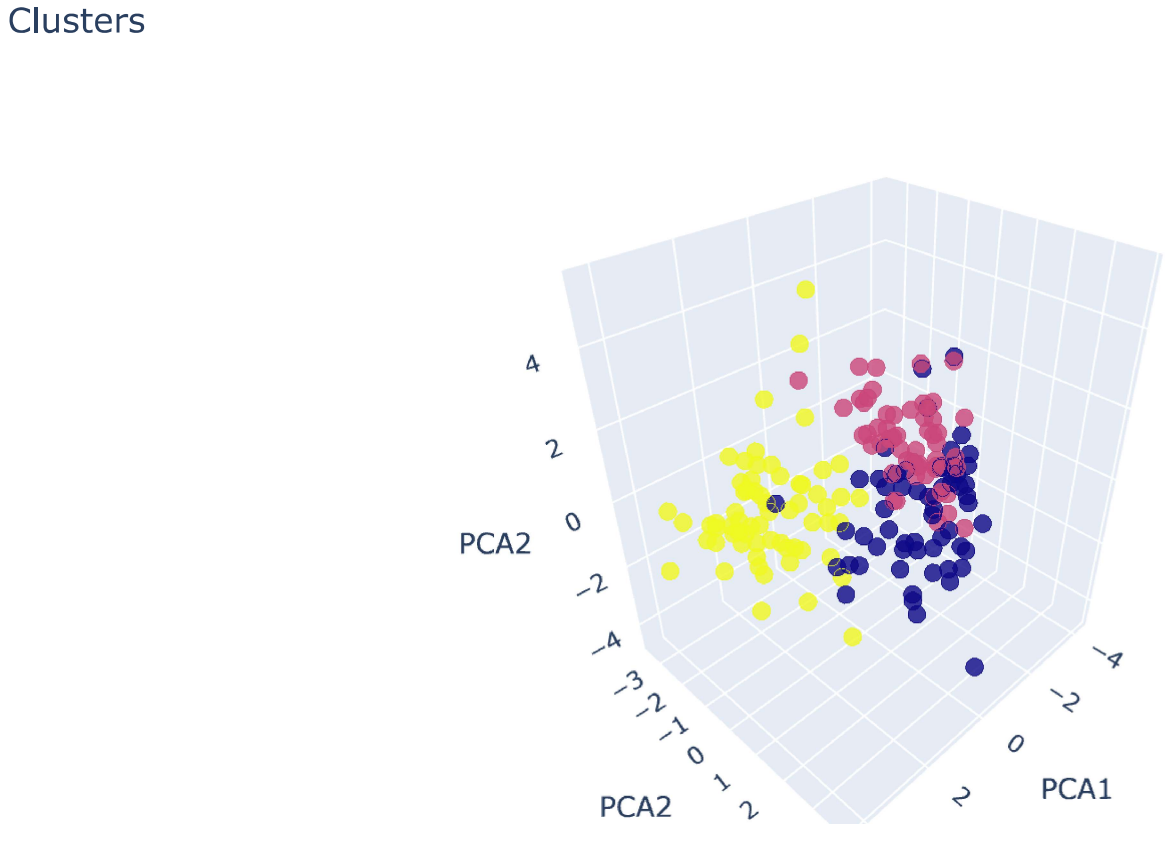
In [32]:
```
from sklearn.cluster import AgglomerativeClustering
HC = AgglomerativeClustering(n_clusters=3, affinity="euclidean", linkage = 'ward')
y_HC = HC.fit_predict(X_)
clusters2 = list(y_HC)
```

In [33]:
```
df_pca_kmeans['Clusters2'] = clusters2
colors2 = df_pca_kmeans["Clusters2"]
df_pca_kmeans.head(5)
```

Out[33]:

	PCA1	PCA2	PCA3	Clusters	Clusters2
0	3.316751	-1.443463	-0.165739	0	2
1	2.209465	0.333393	-2.026457	0	2
2	2.516740	-1.031151	0.982819	0	2
3	3.757066	-2.756372	-0.176192	0	2
4	1.008908	-0.869831	2.026688	0	2

In [34]:
```
trace1 = go.Scatter3d(
    x= df_pca_kmeans['PCA1'],
    y= df_pca_kmeans['PCA2'],
    z= df_pca_kmeans['PCA3'],
   # z1= crime_data['UrbanPop'],
    mode='markers',
     marker=dict(
        color = colors2,
        size= 5,
        line=dict(
            color= colors2,
            width= 12
        ),
        opacity=0.8
     )
)
data = [trace1]
layout = go.Layout(
    title= 'Clusters',
    scene = dict(
            xaxis = dict(title  = 'PCA1'),
            yaxis = dict(title  = 'PCA2'),
            zaxis = dict(title  = 'PCA2')
        )
)
fig = go.Figure(data=data, layout=layout)
py.offline.iplot(fig)
```



In [35]:
```
plt.scatter(df1,df2,c=colors2, edgecolor='black', alpha=0.8,  cmap=plt.cm.get_cmap("YlOrRd",3),s=300)
plt.colorbar()
```

Out[35]: <matplotlib.colorbar.Colorbar at 0x25536548880>