Customer Personality Analysis

In [1]:
```
# Need to perform clustering to summarize customer segments.
#
# The Company can analyze which customer segment is most likely
# to buy the product and then market the product only on that particular segment.
```

In [2]:
```
# The project has the folloing construction

# 1. Data Parsing - processing of the raw data (features optimization, transformatio
# 2. Clusterization of customers
#    2.1 Definition of metrics - silhouette analysis
#    2.2 Standartization of features
#    2.3 Clasterization by KNN method: the final separation to 6 segments
#    2.4 Clasterization by Mini-Batch K-means method: the final separation to 4 segm
#    2.5 Clustering by Affinity Propagation method: the result is pure
#    2.6 Clustering by Agglomerative Агломеративной clustering: the result is worst

# In conclusion, the clustering in 4 segments by KNN or in 6 segments by Mini-Batch
# Further analysis should be done (customers statistics for each of the clustering)
```

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime

from sklearn.svm import SVR

import warnings

%matplotlib inline
```

In [4]:
```
####### People
#ID: Customer's unique identifier
#Year_Birth: Customer's birth year
#Education: Customer's education level
#Marital_Status: Customer's marital status
#Income: Customer's yearly household income
#Kidhome: Number of children in customer's household
#Teenhome: Number of teenagers in customer's household
#Dt_Customer: Date of customer's enrollment with the company
#Recency: Number of days since customer's last purchase
#Complain: 1 if the customer complained in the last 2 years, 0 otherwise

####### Products
#MntWines: Amount spent on wine in last 2 years
#MntFruits: Amount spent on fruits in last 2 years
#MntMeatProducts: Amount spent on meat in last 2 years
#MntFishProducts: Amount spent on fish in last 2 years
#MntSweetProducts: Amount spent on sweets in last 2 years
#MntGoldProds: Amount spent on gold in last 2 years

####### Promotion
#NumDealsPurchases: Number of purchases made with a discount
#AcceptedCmp1: 1 if customer accepted the offer in the 1st campaign, 0 otherwise
#AcceptedCmp2: 1 if customer accepted the offer in the 2nd campaign, 0 otherwise
```

```
#AcceptedCmp3: 1 if customer accepted the offer in the 3rd campaign, 0 otherwise
#AcceptedCmp4: 1 if customer accepted the offer in the 4th campaign, 0 otherwise
#AcceptedCmp5: 1 if customer accepted the offer in the 5th campaign, 0 otherwise
#Response: 1 if customer accepted the offer in the last campaign, 0 otherwise

####### Place
#NumWebPurchases: Number of purchases made through the company’s website
#NumCatalogPurchases: Number of purchases made using a catalogue
#NumStorePurchases: Number of purchases made directly in stores
#NumWebVisitsMonth: Number of visits to company’s website in the last month
```

In [5]:

```
df = pd.read_csv('Data/marketing_campaign.csv', sep='\t', skipinitialspace=True)
```

In [6]:

```
df.head()
```

Out[6]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recenc
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	5
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	3
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	2
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	2
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	9

5 rows × 29 columns

1. Data Parsing

In [7]:

```
df.describe()
```

Out[7]:

	ID	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWine
count	2240.000000	2240.000000	2216.000000	2240.000000	2240.000000	2240.000000	2240.00000
mean	5592.159821	1968.805804	52247.251354	0.444196	0.506250	49.109375	303.93571
std	3246.662198	11.984069	25173.076661	0.538398	0.544538	28.962453	336.59739
min	0.000000	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.00000
25%	2828.250000	1959.000000	35303.000000	0.000000	0.000000	24.000000	23.75000
50%	5458.500000	1970.000000	51381.500000	0.000000	0.000000	49.000000	173.50000
75%	8427.750000	1977.000000	68522.000000	1.000000	1.000000	74.000000	504.25000
max	11191.000000	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493.00000

8 rows × 26 columns

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
 19  NumWebVisitsMonth    2240 non-null   int64  
 20  AcceptedCmp3         2240 non-null   int64  
 21  AcceptedCmp4         2240 non-null   int64  
 22  AcceptedCmp5         2240 non-null   int64  
 23  AcceptedCmp1         2240 non-null   int64  
 24  AcceptedCmp2         2240 non-null   int64  
 25  Complain             2240 non-null   int64  
 26  Z_CostContact        2240 non-null   int64  
 27  Z_Revenue            2240 non-null   int64  
 28  Response             2240 non-null   int64  
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

1.1 Transform the column "Dt_Customer" to the time since the customer is registered

In [9]:

```
# first transform the column to datetime format
df['Dt_Customer'] = pd.to_datetime(df['Dt_Customer'],format='%d-%m-%Y')
```

In [10]:

```
# then calculate the time of existing each customer on the web-site in months
max_date = datetime(2014, 10, 4)
df['presence'] = df.apply(lambda x: (max_date - x['Dt_Customer']).days/30.0, axis =
```

1.2 Optimize the features

1.2.1 Education

In [11]:

```
df['Education'].unique()
```

Out[11]:

```
array(['Graduation', 'PhD', 'Master', 'Basic', '2n Cycle'], dtype=object)
```

In [12]:

```
df['Education'] = df['Education'].replace({'Basic':'Undergraduate','2n Cycle':'Under
```

In [13]:

```
df['Education'].unique()
```

Out[13]:
```
array(['Postgraduate', 'PhD', 'Undergraduate'], dtype=object)
```

1.2.2 Marital_Status

In [14]:
```
df['Marital_Status'].unique()
```

Out[14]:
```
array(['Single', 'Together', 'Married', 'Divorced', 'Widow', 'Alone',
       'Absurd', 'YOLO'], dtype=object)
```

In [15]:
```
#widow - вдова
#yolo = absurd
#together = married
#alone = single
```

In [16]:
```
df['Marital_Status'] = df['Marital_Status'].replace({'Divorced':'Alone','Single':'Al
```

In [17]:
```
df['Marital_Status'].unique()
```

Out[17]:
```
array(['Alone', 'Together'], dtype=object)
```

1.2.3 Year_Birth

In [18]:
```
# it is more convenient to deal with age, not year of birth
df['Age'] = 2014 - df['Year_Birth']
```

1.2.4 Keedhome & Teenhome

In [19]:
```
# calculate the total number of children
df['Children'] = df['Kidhome'] + df['Teenhome']
```

1.2.5 MntWines & MntFruits & MntMeatProducts & MntFishProducts & MntSweetProducts & MntGoldProds

In [20]:
```
# calculate total spending
df['Spending'] = df['MntWines']+df['MntFruits']+df['MntMeatProducts']+df['MntFishPro
```

1.2.6 Rename some features name for simplicity

In [21]:
```
df = df.rename(columns={'MntWines': "Wines",'MntFruits':'Fruits','MntMeatProducts':'
```

1.2.7 Keep only selected features for further analysis

In [22]:
```
data=df[['Age','Education','Marital_Status','Income','Spending','presence','Children
```

In [23]:
```
data.head()
```

Out[23]:

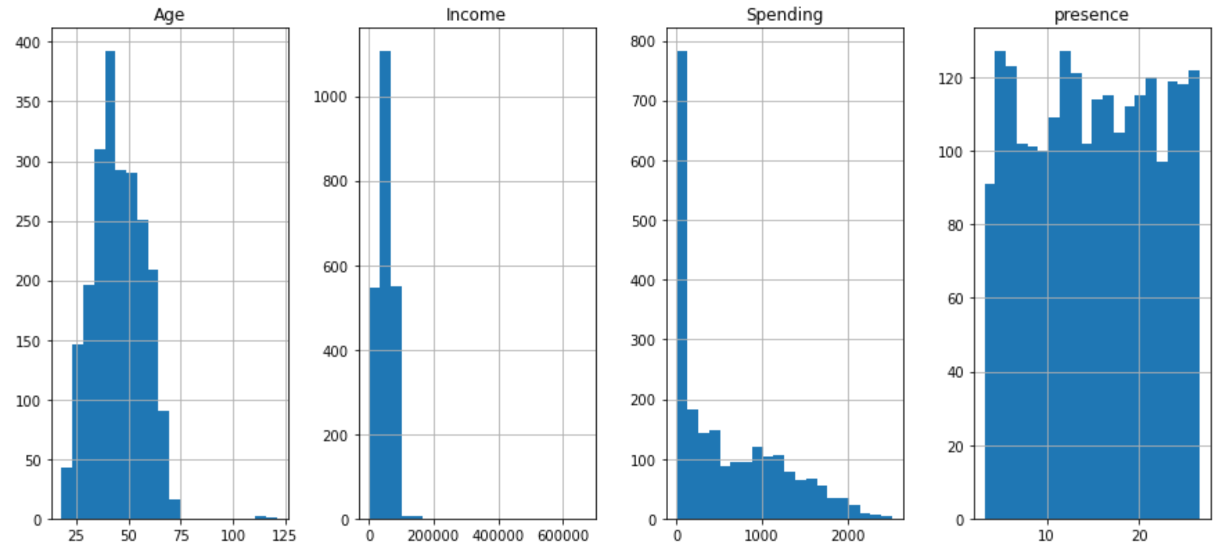
	Age	Education	Marital_Status	Income	Spending	presence	Children
0	57	Postgraduate	Alone	58138.0	1617	25.333333	0
1	60	Postgraduate	Alone	46344.0	27	7.000000	2
2	49	Postgraduate	Together	71613.0	776	13.633333	0
3	30	Postgraduate	Together	26646.0	53	7.866667	1

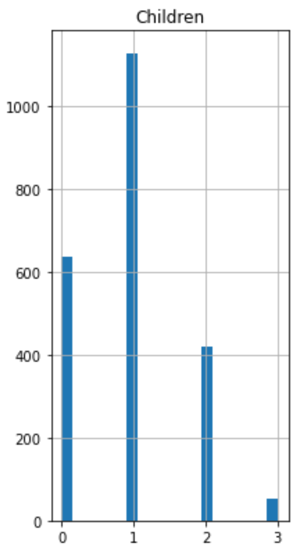
	Age	Education	Marital_Status	Income	Spending	presence	Children
4	33	PhD	Together	58293.0	422	8.600000	1

1.3 Remove NaN values and outliers

In [24]:

```
# First look at the statictics
data.hist(bins=20, figsize=(15,15),layout=(2,4));
```





In [25]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             2240 non-null   int64  
 1   Education       2240 non-null   object 
 2   Marital_Status  2240 non-null   object 
 3   Income          2216 non-null   float64
 4   Spending        2240 non-null   int64  
 5   presence        2240 non-null   float64
 6   Children        2240 non-null   int64  
```

```
dtypes: float64(2), int64(3), object(2)
memory usage: 122.6+ KB
```

In [26]:

```
# Income feature is very important and there is only one feature with NaN and outlie
# Delete all such raws, where Income either outlier or NaN (there are only 24 such r
```

In [27]:

```
data = data.dropna(subset=['Income'])
data = data[data['Income'] < 600000]
```

In [28]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2215 entries, 0 to 2239
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             2215 non-null   int64  
 1   Education       2215 non-null   object 
 2   Marital_Status  2215 non-null   object 
 3   Income          2215 non-null   float64
 4   Spending        2215 non-null   int64  
 5   presence        2215 non-null   float64
 6   Children        2215 non-null   int64  
dtypes: float64(2), int64(3), object(2)
memory usage: 138.4+ KB
```

In [29]:

```
### 1.4 Transform nonnumeric features 'Education' and 'Marital_Status' to the set of
```

In [30]:

```
list(data['Education'].unique())
```

Out[30]:

```
['Postgraduate', 'PhD', 'Undergraduate']
```

In [31]:

```
# functions of creation of new features - educations
def Edu1(education):
    if education == 'Postgraduate':
        return 1
    else:
        return 0
def Edu2(education):
    if education == 'PhD':
        return 1
    else:
        return 0
def Edu3(education):
    if education == 'Undergraduate':
        return 1
    else:
        return 0
```

In [32]:

```
data['PostGr'] = data.apply(lambda x: Edu1(x['Education']), axis = 1)
data['PhD'] = data.apply(lambda x: Edu2(x['Education']), axis = 1)
data['UnderGr'] = data.apply(lambda x: Edu3(x['Education']), axis = 1)
```

In [33]:

```
list(data['Marital_Status'].unique())
```

Out[33]: ['Alone', 'Together']

In [34]:
```
# functions of creation of new features - family status
def Mar1(status):
    if status == 'Alone':
        return 1
    else:
        return 0
def Mar2(status):
    if status == 'Together':
        return 1
    else:
        return 0
```

In [35]:
```
data['Alone'] = data.apply(lambda x: Mar1(x['Marital_Status']), axis = 1)
data['Together'] = data.apply(lambda x: Mar2(x['Marital_Status']), axis = 1)
```

In [36]:
```
### 1.5 Select features for further clusterization
```

In [37]:
```
data_a = data[['Age','Income','Spending','presence','Children','PostGr','PhD','Under
```

In [38]:
```
data_a
```

Out[38]:

	Age	Income	Spending	presence	Children	PostGr	PhD	UnderGr	Alone	Together
0	57	58138.0	1617	25.333333	0	1	0	0	1	0
1	60	46344.0	27	7.000000	2	1	0	0	1	0
2	49	71613.0	776	13.633333	0	1	0	0	0	1
3	30	26646.0	53	7.866667	1	1	0	0	0	1
4	33	58293.0	422	8.600000	1	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...
2235	47	61223.0	1341	15.933333	1	1	0	0	0	1
2236	68	64014.0	444	3.866667	3	0	1	0	0	1
2237	33	56981.0	1241	8.400000	0	1	0	0	1	0
2238	58	69245.0	843	8.433333	1	1	0	0	0	1
2239	60	52869.0	172	23.966667	2	0	1	0	0	1

2215 rows × 10 columns

2. Clustering

In [39]:
```
# The main problem is an appraising the optimal number of clusters.

# I have decided to make this appraising by silhouette analysis

# The core of this analysis is calculation of spacial destance between the clusters.
# The final silhouette graph displays a vicinity of each point of one cluster to the
```

```
# and thus provides the visual means of appraising of the number of clusters.
# The final measure has the range [-1, 1].

# The silhouette coefficients near +1 shows that the point is far away from the othe
# 0 - the point is on the border or close to other cluster, negative values show the

# Also the cluster size can be visualized based on its silhouette width on the graph
```

2.1 Metrics of quality for clustering

In [40]:

```
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
```

In [41]:

```
def sil_plot(model, data, n_clusters):
    result = model.fit_predict(data)
    sil = silhouette_samples(data, result)
    silhouette_avg = silhouette_score(data, result)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :{:.3f}".format(silhouette_avg))
    y_lower = 10

    fig, ax = plt.subplots()
    fig.set_size_inches(18, 7)

    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.2, 1]
    ax.set_xlim([-0.2, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax.set_ylim([0, len(data) + (n_clusters + 1) * 10])

    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = sil[result == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        cmap = cm.get_cmap("Spectral")
        color = cmap(float(i) / n_clusters)

        ax.fill_betweenx(np.arange(y_lower, y_upper),
                         0, ith_cluster_silhouette_values,
                         facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    # The vertical line for average silhouette score of all the values
    ax.axvline(x=silhouette_avg, color="red", linestyle="--")
    ax.set_yticks([])  # Clear the yaxis labels / ticks
    ax.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
    ax.set_xlabel("The silhouette coefficient values")
    ax.set_ylabel("Cluster label")
    plt.suptitle(("Silhouette analysis for clustering on sample data "
```

```
                    "with n_clusters = %d" % n_clusters),
                   fontsize=14, fontweight='bold')
```

2.2 Scaling of the features

In [42]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data = pd.DataFrame(scaler.fit_transform(data_a), columns = data_a.columns, i
scaled_data.head()
```

Out[42]:

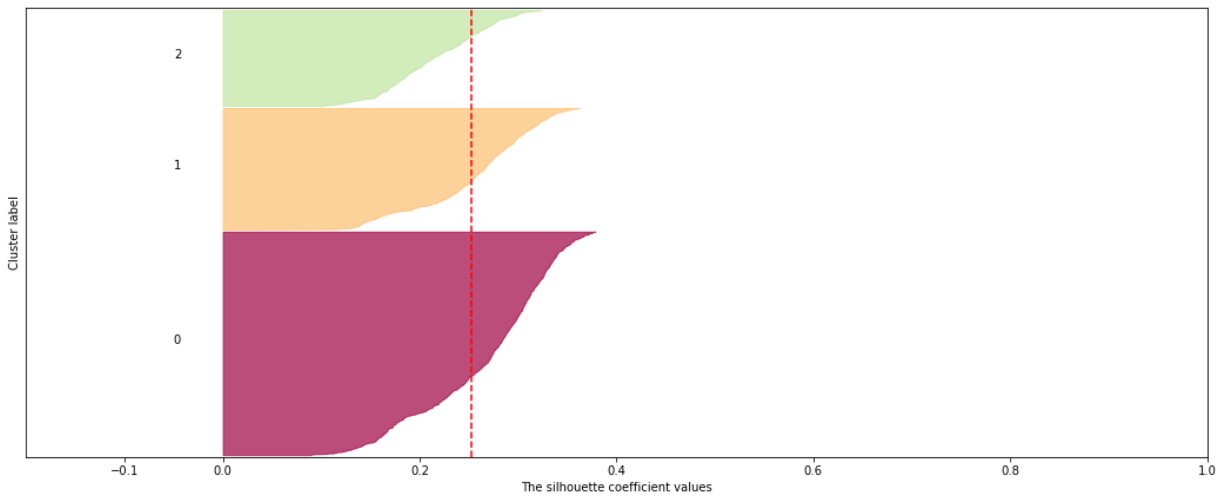
	Age	Income	Spending	presence	Children	PostGr	PhD	UnderGr	Alone
0	0.986016	0.286604	1.675011	1.528882	-1.264487	0.704714	-0.526681	-0.359897	1.348357
1	1.236344	-0.261407	-0.962727	-1.188066	1.405522	0.704714	-0.526681	-0.359897	1.348357
2	0.318476	0.912723	0.279830	-0.205025	-1.264487	0.704714	-0.526681	-0.359897	-0.741643
3	-1.266933	-1.176680	-0.919594	-1.059629	0.070517	0.704714	-0.526681	-0.359897	-0.741643
4	-1.016605	0.293806	-0.307440	-0.950951	0.070517	-1.419016	1.898681	-0.359897	-0.741643

2.3 KNN method

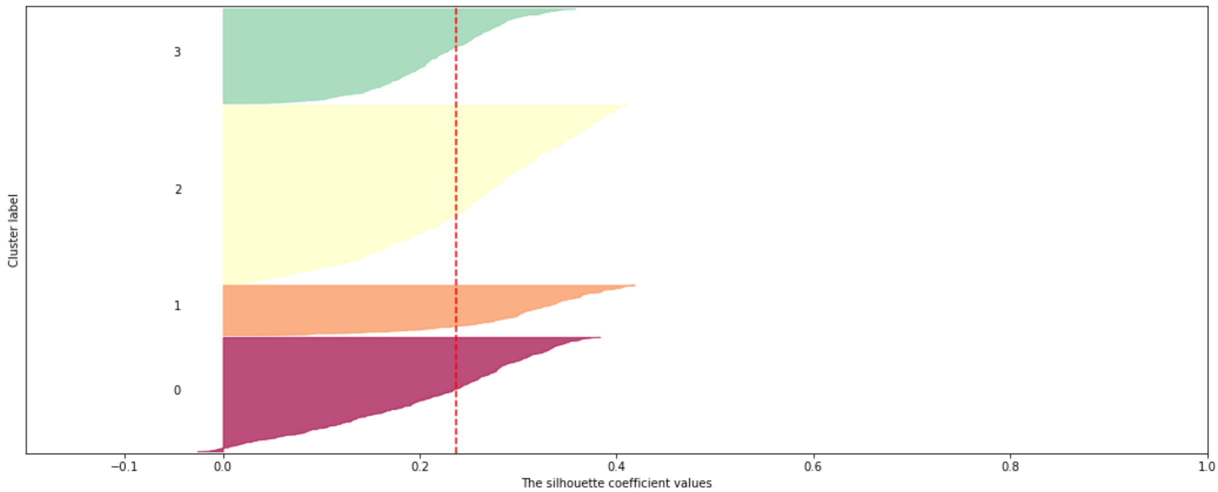
In [43]:
```
# try to run on # of clusters from 3 to 15, appraising the result by silhouettes
for n in range(3,16):
    KMN = KMeans(n_clusters=n, random_state=1)
    sil_plot(KMN, scaled_data, n)
```

```
For n_clusters = 3 The average silhouette_score is :0.253
For n_clusters = 4 The average silhouette_score is :0.237
For n_clusters = 5 The average silhouette_score is :0.251
For n_clusters = 6 The average silhouette_score is :0.280
For n_clusters = 7 The average silhouette_score is :0.285
For n_clusters = 8 The average silhouette_score is :0.287
For n_clusters = 9 The average silhouette_score is :0.255
For n_clusters = 10 The average silhouette_score is :0.265
For n_clusters = 11 The average silhouette_score is :0.259
For n_clusters = 12 The average silhouette_score is :0.262
For n_clusters = 13 The average silhouette_score is :0.241
For n_clusters = 14 The average silhouette_score is :0.244
For n_clusters = 15 The average silhouette_score is :0.243
```

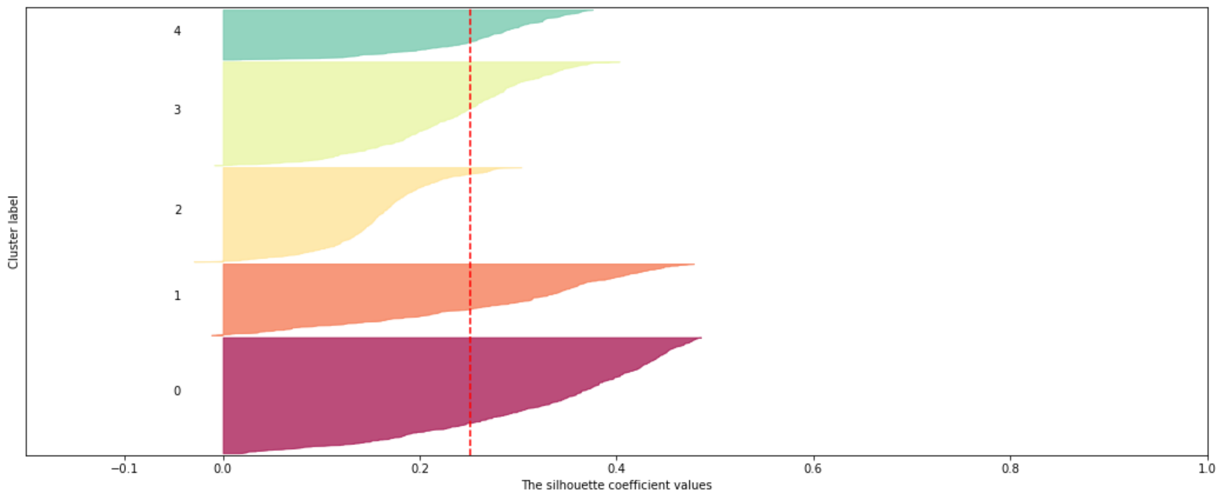
Silhouette analysis for clustering on sample data with n_clusters = 3

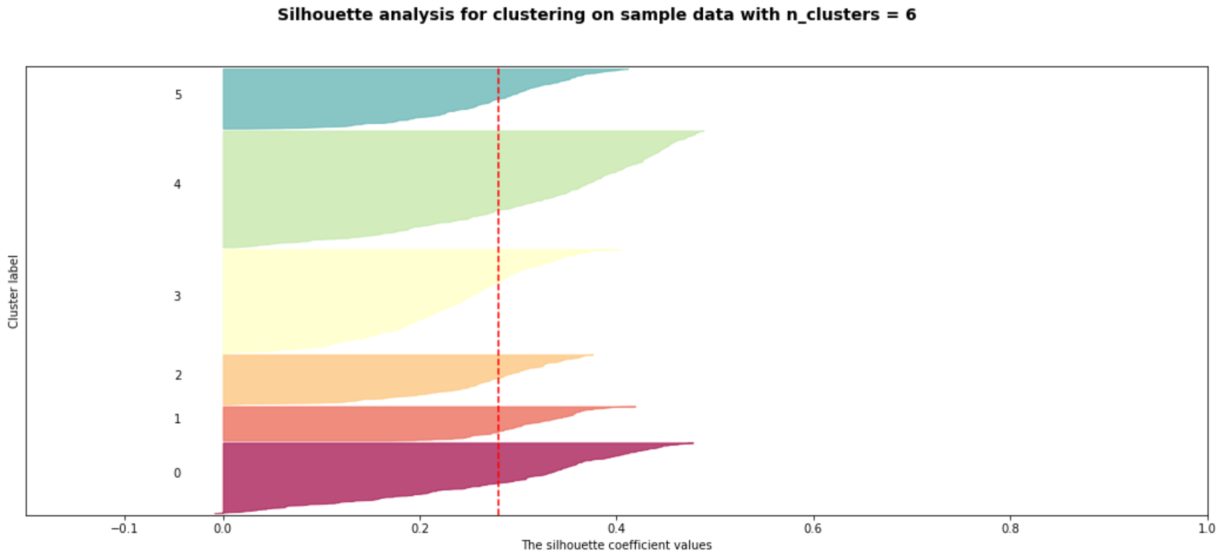


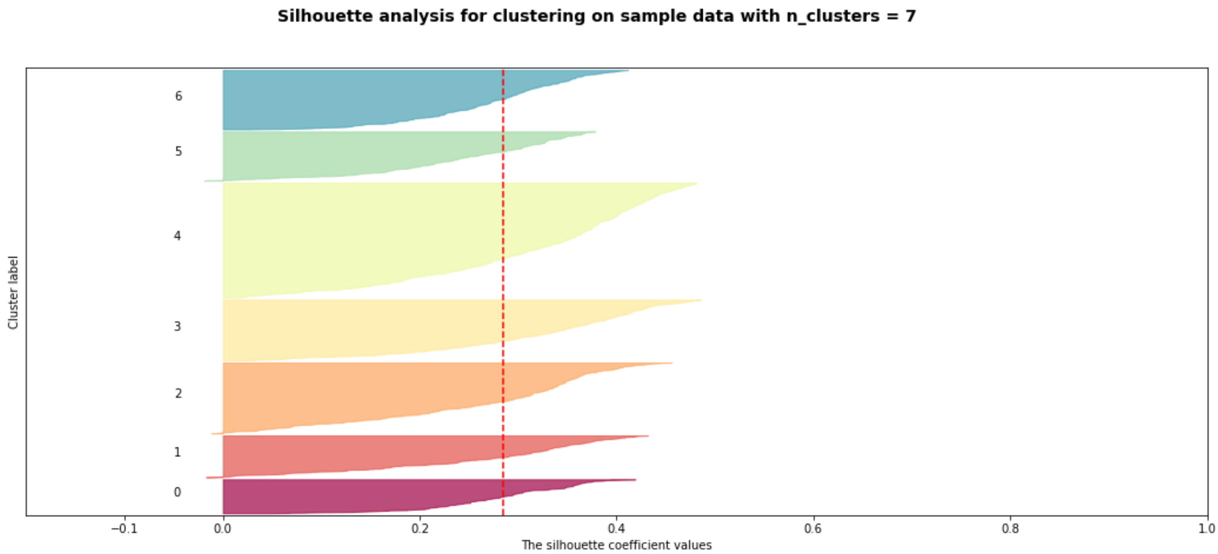
Silhouette analysis for clustering on sample data with n_clusters = 4

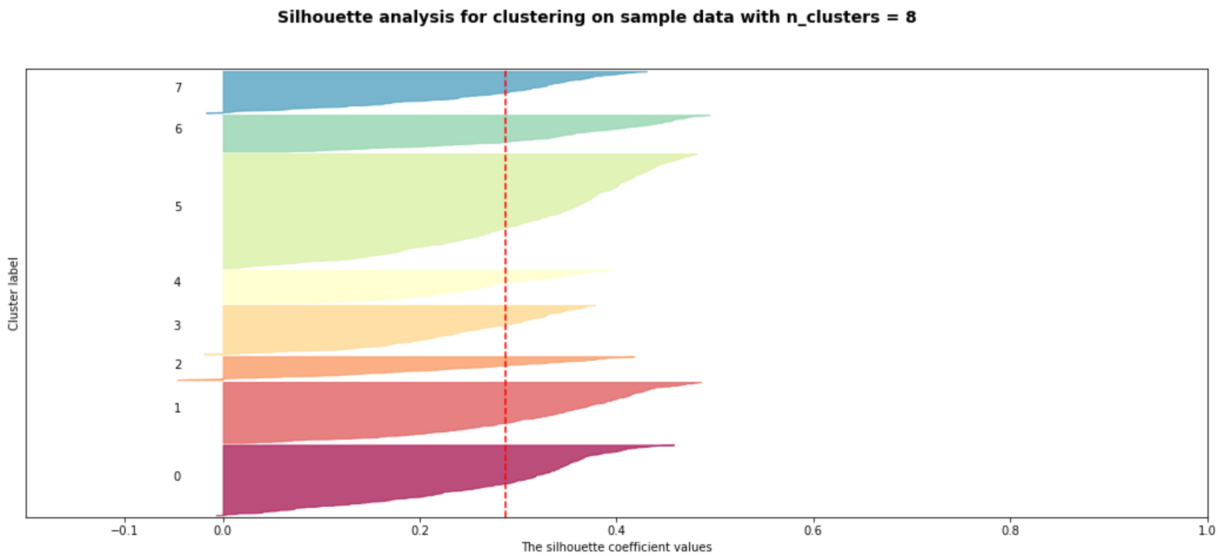


Silhouette analysis for clustering on sample data with n_clusters = 5

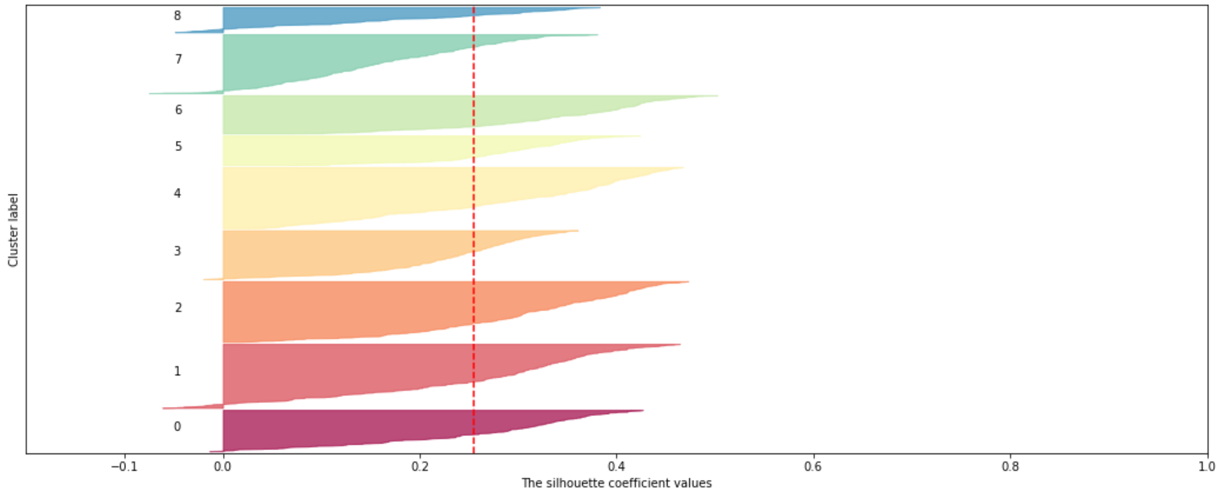



Silhouette analysis for clustering on sample data with n_clusters = 6
Cluster label
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

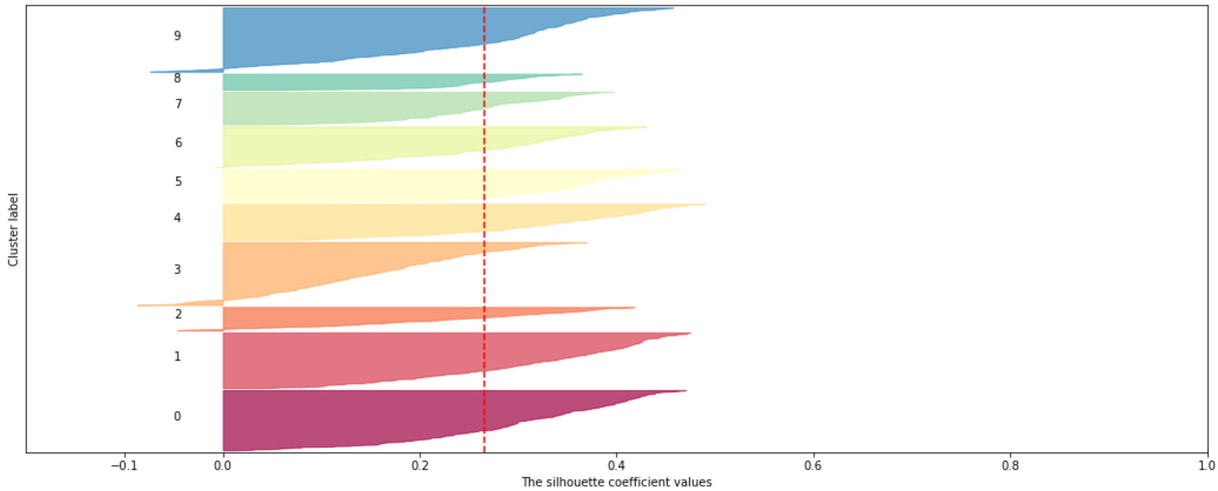

Silhouette analysis for clustering on sample data with n_clusters = 7
Cluster label
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


Silhouette analysis for clustering on sample data with n_clusters = 8
Cluster label
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

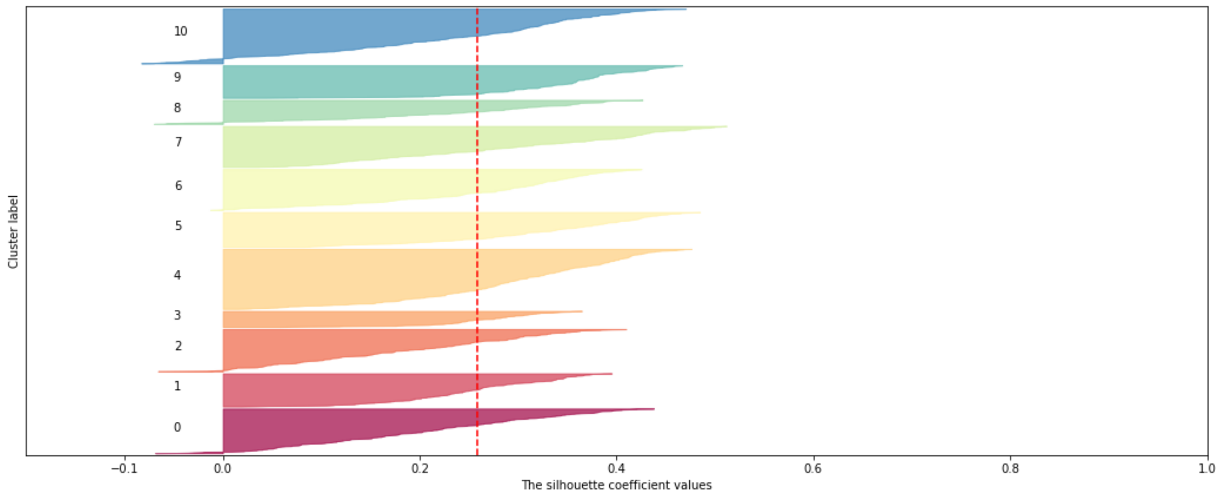
Silhouette analysis for clustering on sample data with n_clusters = 9



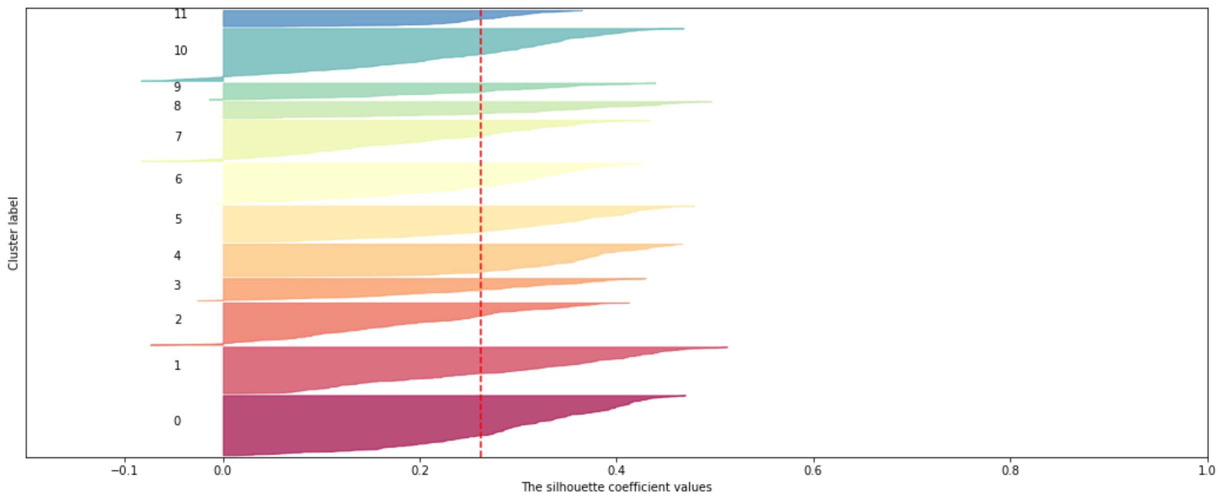
Silhouette analysis for clustering on sample data with n_clusters = 10



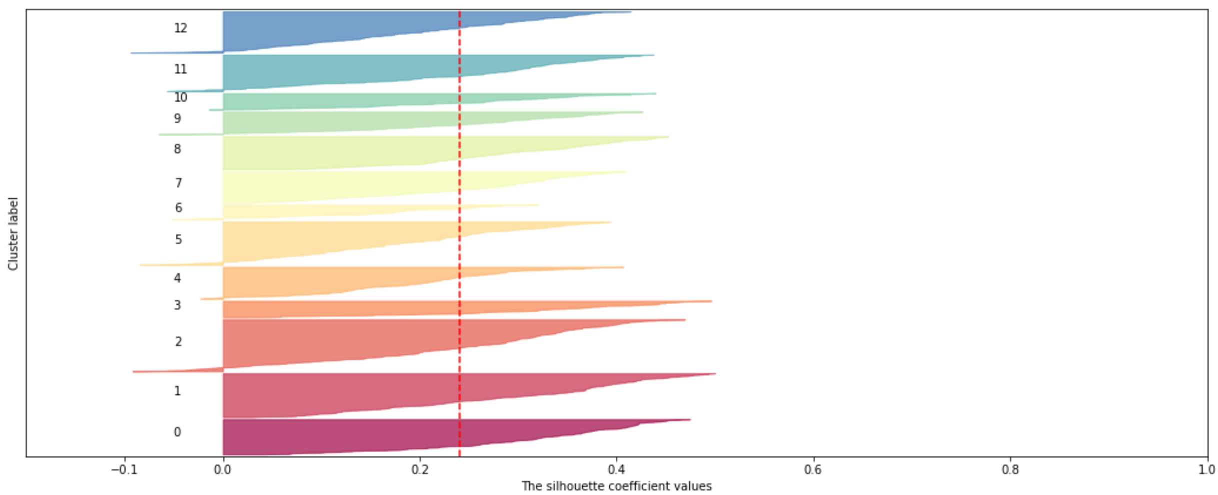
Silhouette analysis for clustering on sample data with n_clusters = 11



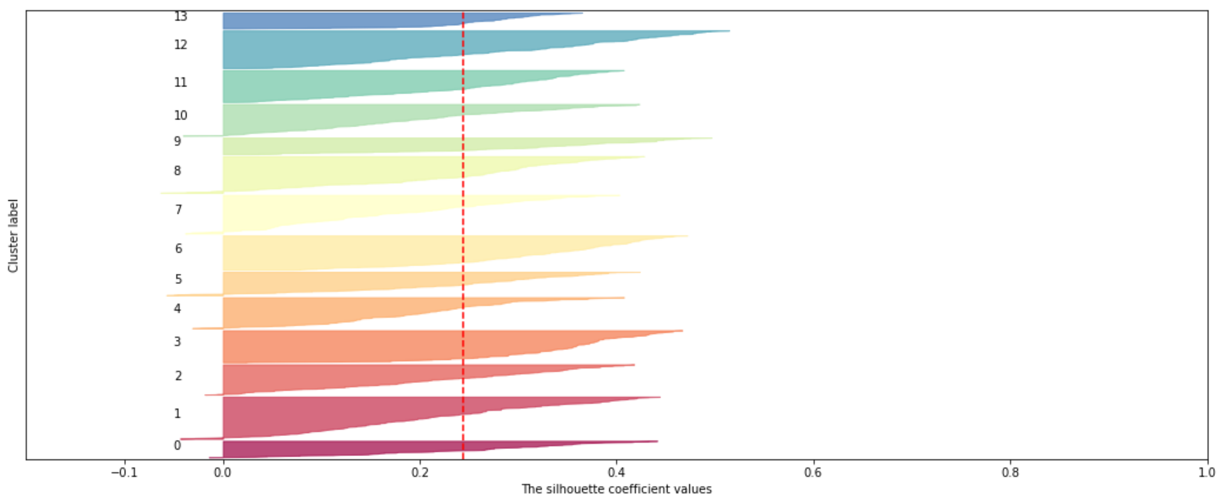
Silhouette analysis for clustering on sample data with n_clusters = 12

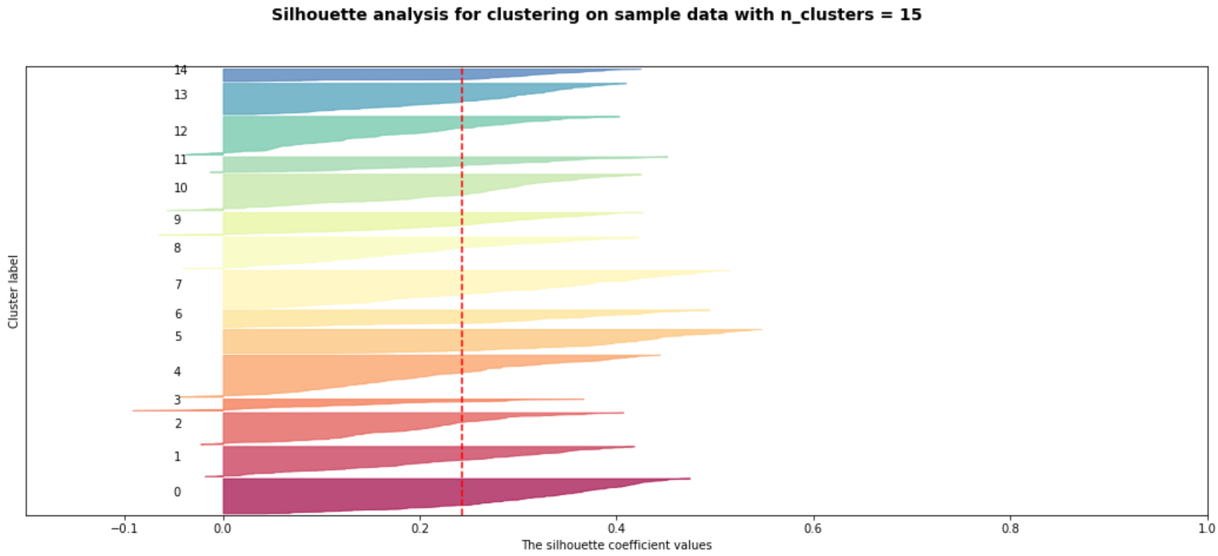


Silhouette analysis for clustering on sample data with n_clusters = 13



Silhouette analysis for clustering on sample data with n_clusters = 14





In [44]:
```
# Visually the best result is for 3 and 6 clusters
# Herewith, the average silhouette coefficient for 6 clusters is bigger. So, stop on

KNM = KMeans(n_clusters=6, random_state=1)
resultKNM = KNM.fit_predict(scaled_data)
# number of customers within each cluster for 6-clustering
np.unique(resultKNM, return_counts=True)
```

Out[44]:
```
(array([0, 1, 2, 3, 4, 5]), array([360, 175, 254, 526, 595, 305], dtype=int64))
```

In [45]:
```
print("the average spending in 1-st segment is: {:.0f}".format(data_a.loc[scaled_dat
print("the average spending in 2-nd segment is: {:.0f}".format(data_a.loc[scaled_dat
print("the average spending in 3-d segment is: {:.0f}".format(data_a.loc[scaled_data
print("the average spending in 4-th segment is: {:.0f}".format(data_a.loc[scaled_dat
print("the average spending in 5-th segment is: {:.0f}".format(data_a.loc[scaled_dat
print("the average spending in 6-th segment is: {:.0f}".format(data_a.loc[scaled_dat
```

```
the average spending in 1-st segment is: 1271
the average spending in 2-nd segment is: 646
the average spending in 3-d segment is: 407
the average spending in 4-th segment is: 636
the average spending in 5-th segment is: 211
the average spending in 6-th segment is: 691
```

In [46]:
```
# The first segment is the most important, as the mean spending is much higher for t
```

2.4 Mini-Batch K-means method

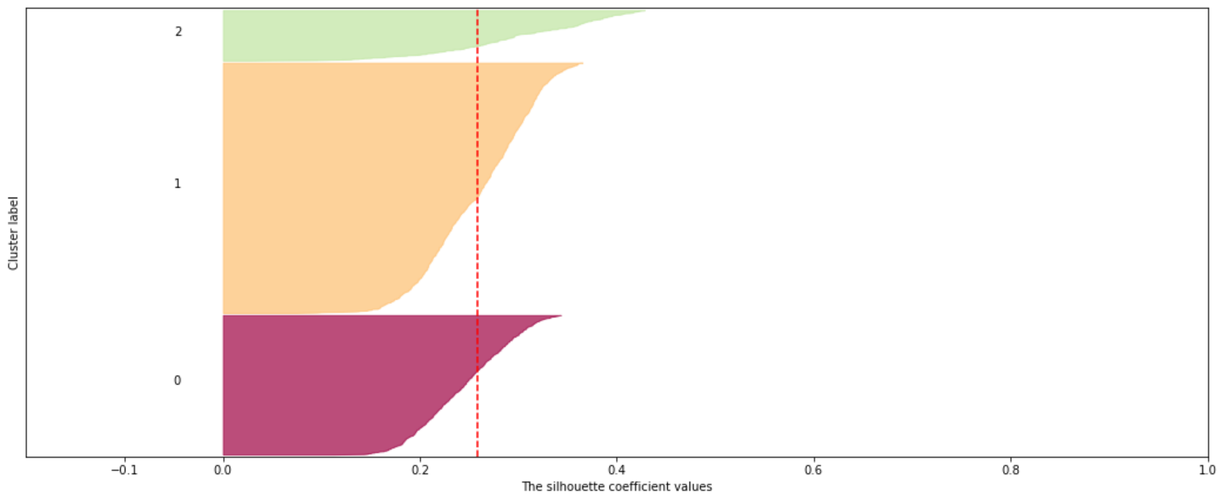
In [47]:
```
from sklearn.cluster import KMeans, MiniBatchKMeans, AffinityPropagation
```

In [48]:
```
# try to run on # of clusters from 3 to 15, appraising the result by silhouettes
warnings.filterwarnings('ignore')
for n in range(3,16):
    MBKM = MiniBatchKMeans(n_clusters=n, random_state=1)
    sil_plot(MBKM, scaled_data, n)
```

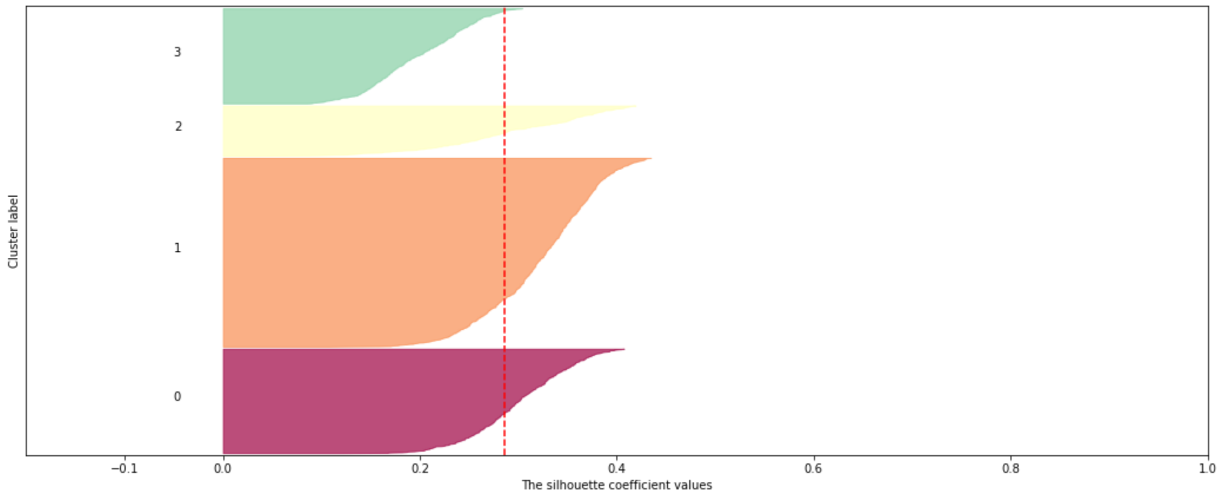
```
For n_clusters = 3 The average silhouette_score is :0.259
For n_clusters = 4 The average silhouette_score is :0.287
For n_clusters = 5 The average silhouette_score is :0.252
For n_clusters = 6 The average silhouette_score is :0.257
```

```
For n_clusters = 7 The average silhouette_score is :0.265
For n_clusters = 8 The average silhouette_score is :0.246
For n_clusters = 9 The average silhouette_score is :0.238
For n_clusters = 10 The average silhouette_score is :0.233
For n_clusters = 11 The average silhouette_score is :0.225
For n_clusters = 12 The average silhouette_score is :0.241
For n_clusters = 13 The average silhouette_score is :0.242
For n_clusters = 14 The average silhouette_score is :0.245
For n_clusters = 15 The average silhouette_score is :0.207
```

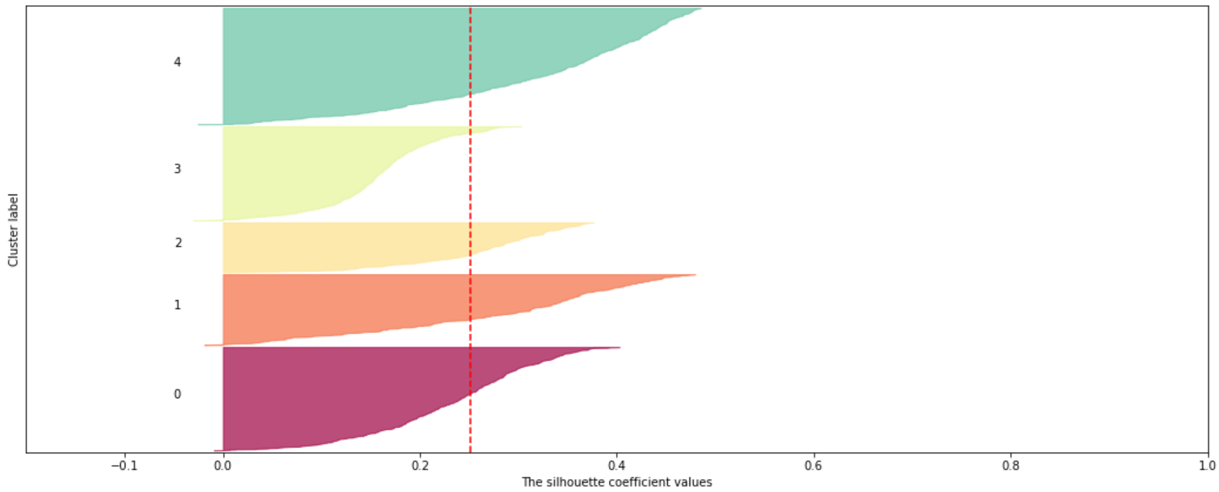
Silhouette analysis for clustering on sample data with n_clusters = 3



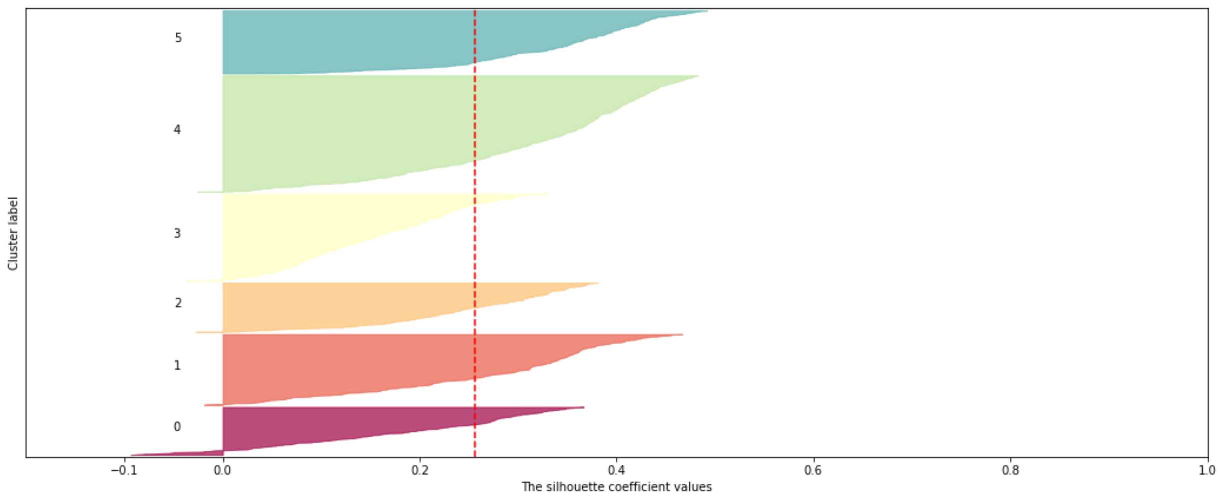
Silhouette analysis for clustering on sample data with n_clusters = 4



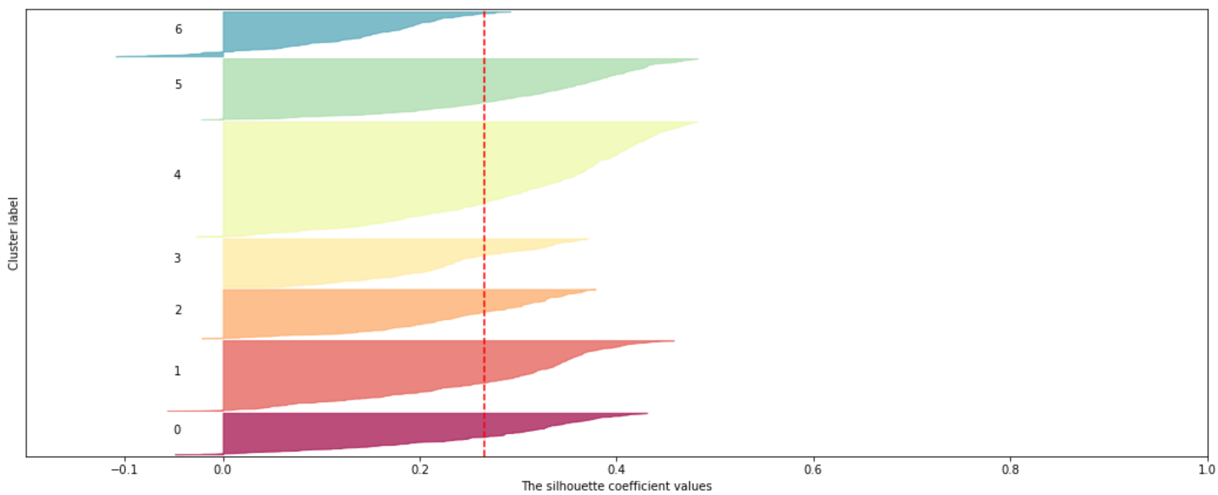
Silhouette analysis for clustering on sample data with n_clusters = 5



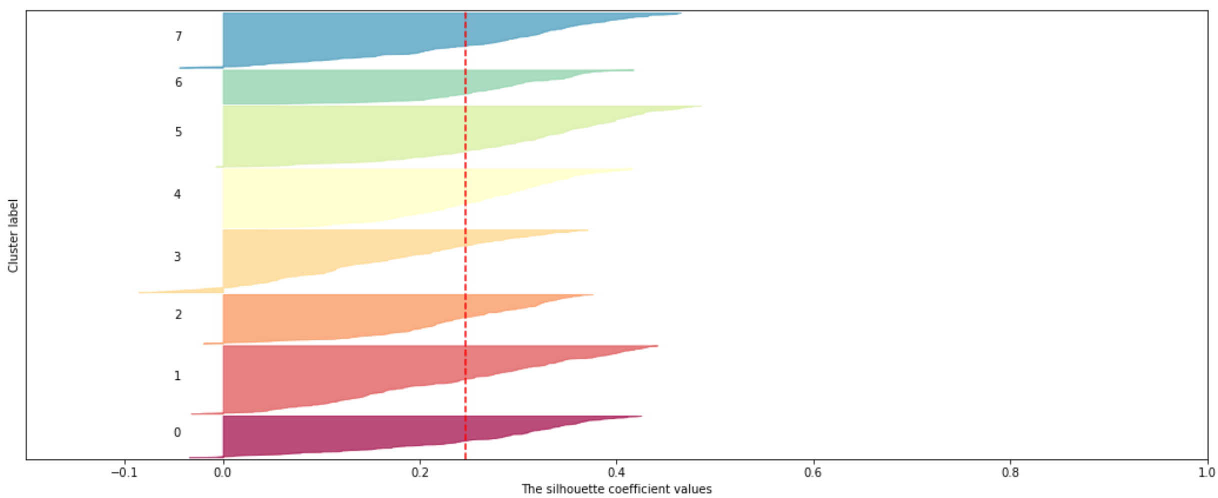
Silhouette analysis for clustering on sample data with n_clusters = 6



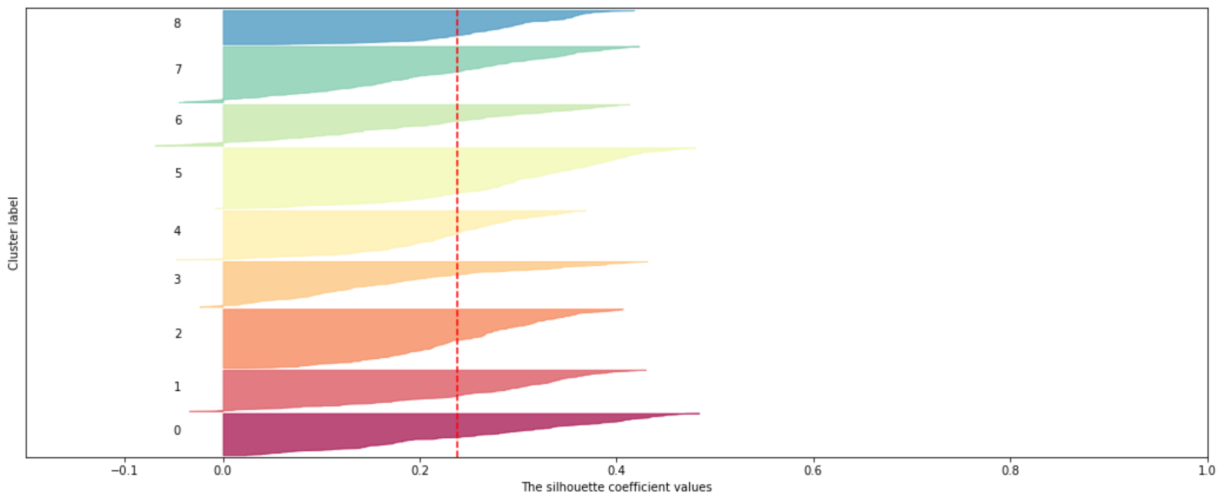
Silhouette analysis for clustering on sample data with n_clusters = 7



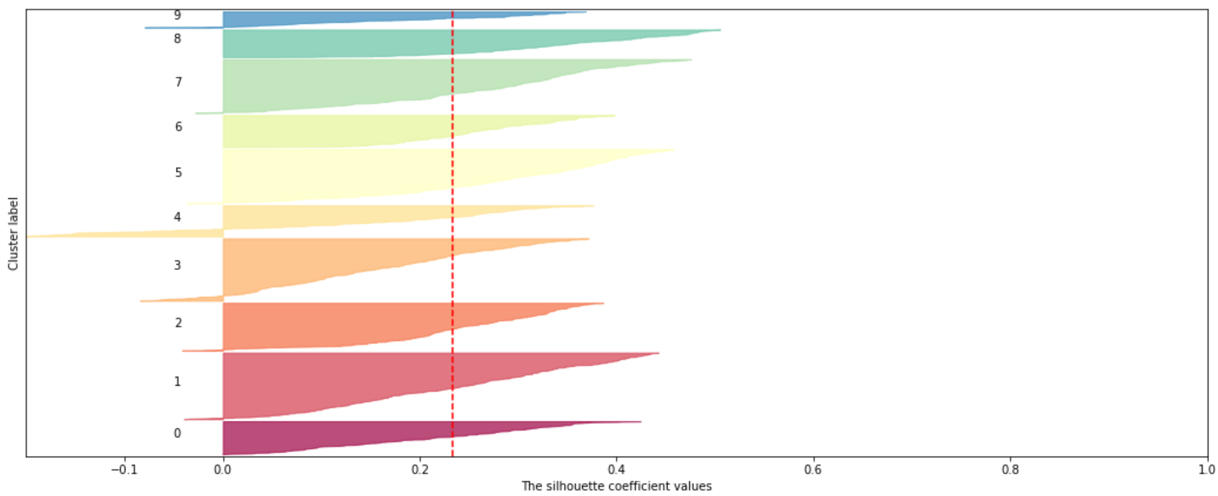
Silhouette analysis for clustering on sample data with n_clusters = 8



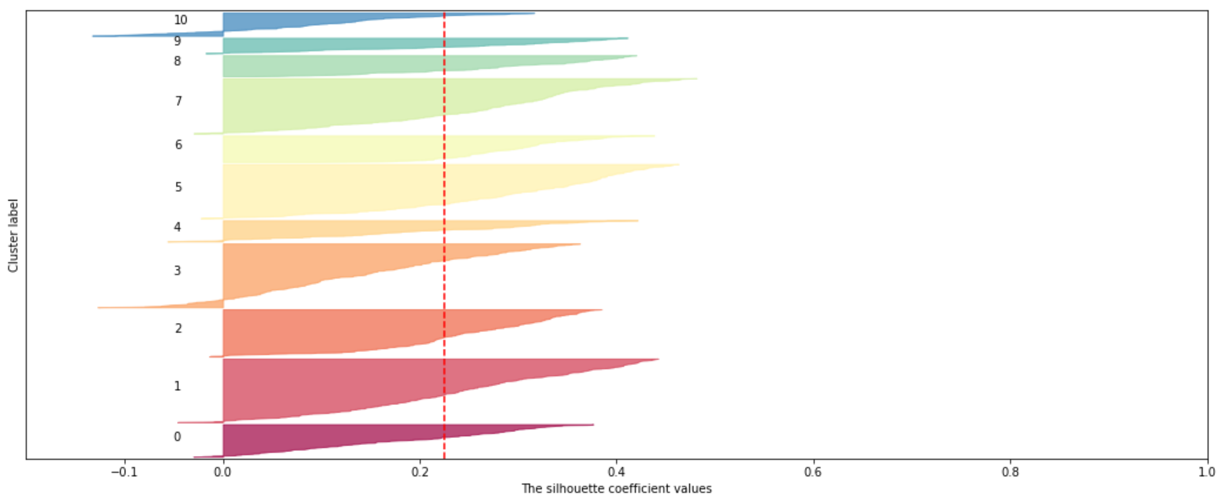
Silhouette analysis for clustering on sample data with n_clusters = 9

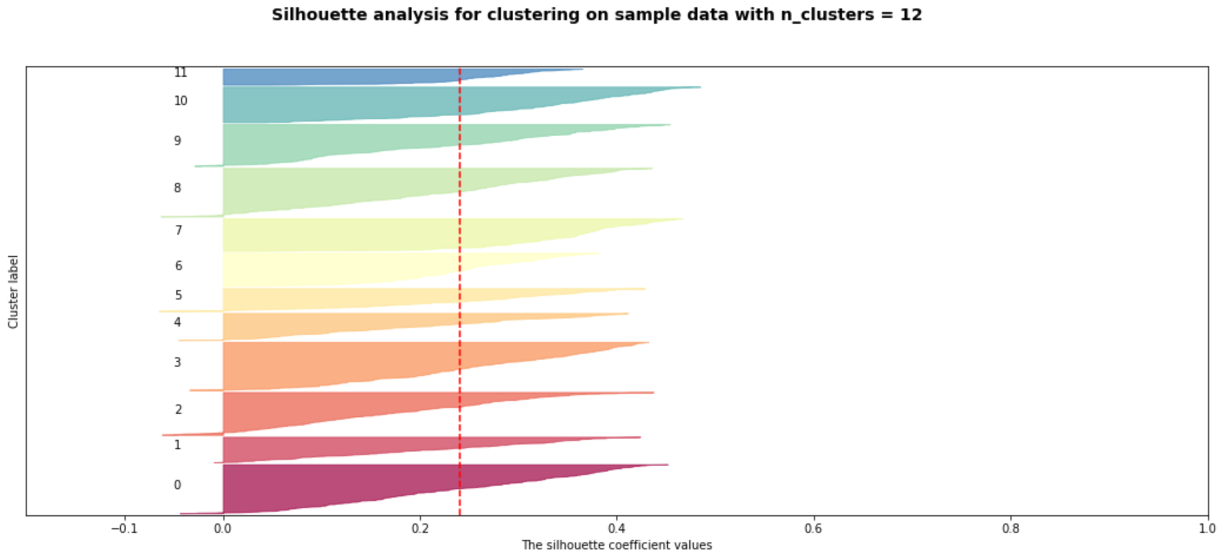


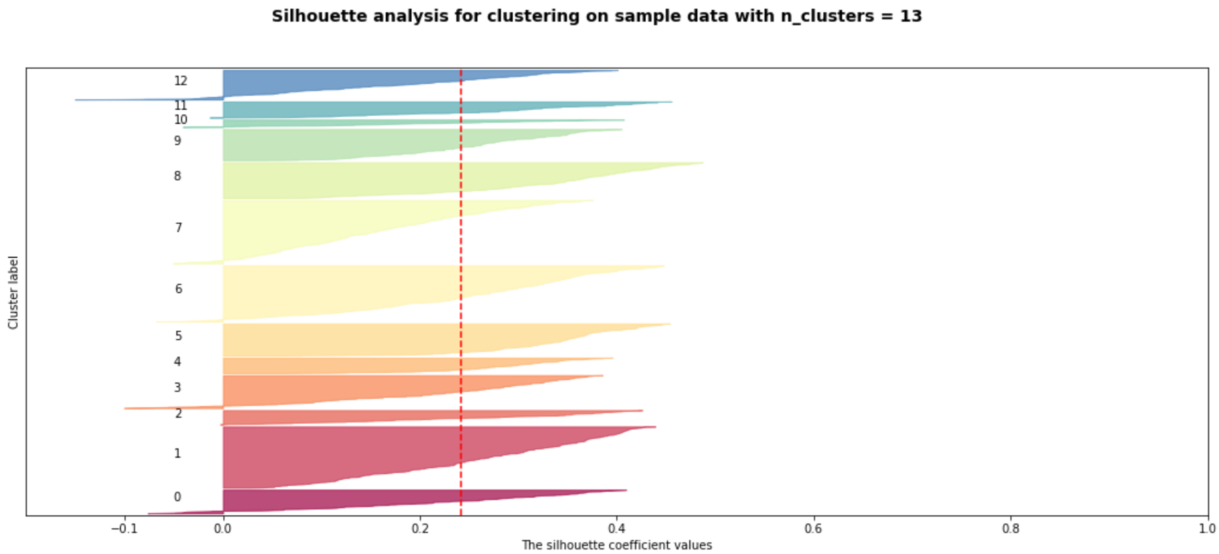
Silhouette analysis for clustering on sample data with n_clusters = 10

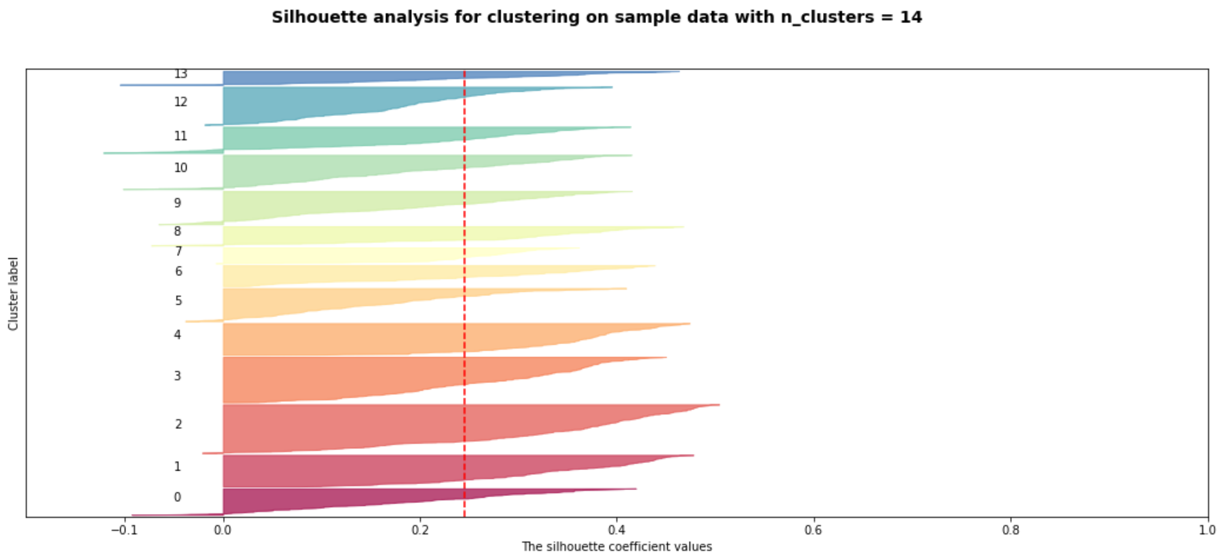


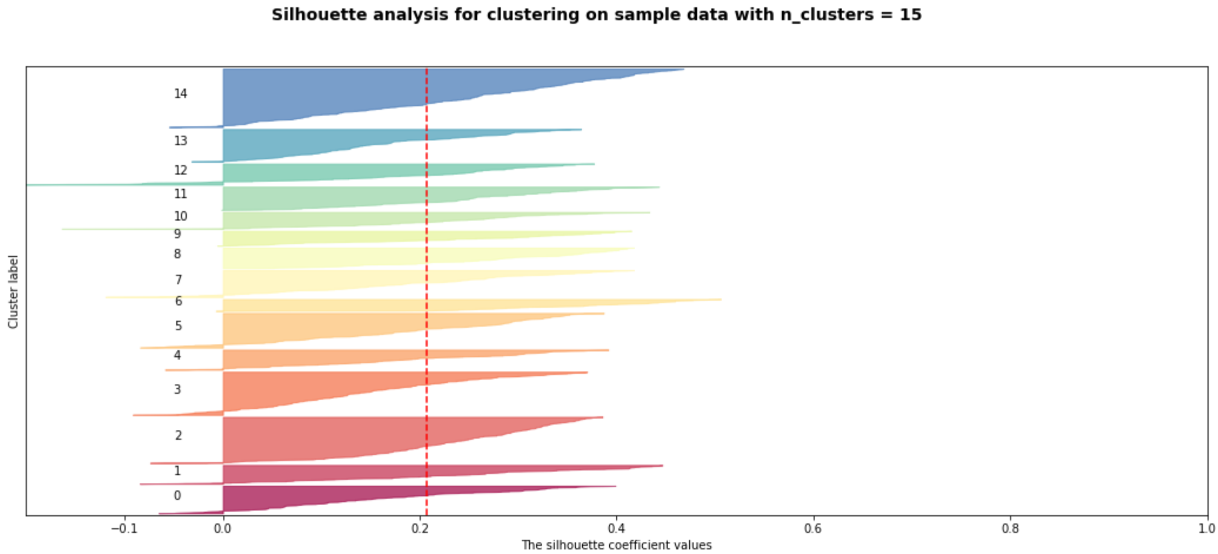
Silhouette analysis for clustering on sample data with n_clusters = 11




Silhouette analysis for clustering on sample data with n_clusters = 12
Cluster label
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
8
9
10
11


Silhouette analysis for clustering on sample data with n_clusters = 13
Cluster label
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
8
9
10
11
12


Silhouette analysis for clustering on sample data with n_clusters = 14
Cluster label
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7
8
9
10
11
12
13



In [49]:
```
# Definitely 4-clustering is the best

MBKM = MiniBatchKMeans(n_clusters=6, random_state=1)
resultMBKM = MBKM.fit_predict(scaled_data)
# number of customers within each cluster for 4-clustering
np.unique(resultMBKM, return_counts=True)
```

Out[49]:
```
(array([0, 1, 2, 3, 4, 5]), array([245, 361, 252, 443, 593, 321], dtype=int64))
```

In [50]:
```
print("the average spending in 1-st segment is: {:.0f}".format(data_a.loc[scaled_dat
print("the average spending in 2-nd segment is: {:.0f}".format(data_a.loc[scaled_dat
print("the average spending in 3-d segment is: {:.0f}".format(data_a.loc[scaled_data
print("the average spending in 4-th segment is: {:.0f}".format(data_a.loc[scaled_dat
```

```
the average spending in 1-st segment is: 1328
the average spending in 2-nd segment is: 1265
the average spending in 3-d segment is: 397
the average spending in 4-th segment is: 609
```

In [51]:
```
# The first two segments are the most important, as the mean spending is much higher
```

2.5 Affinity Propagation method

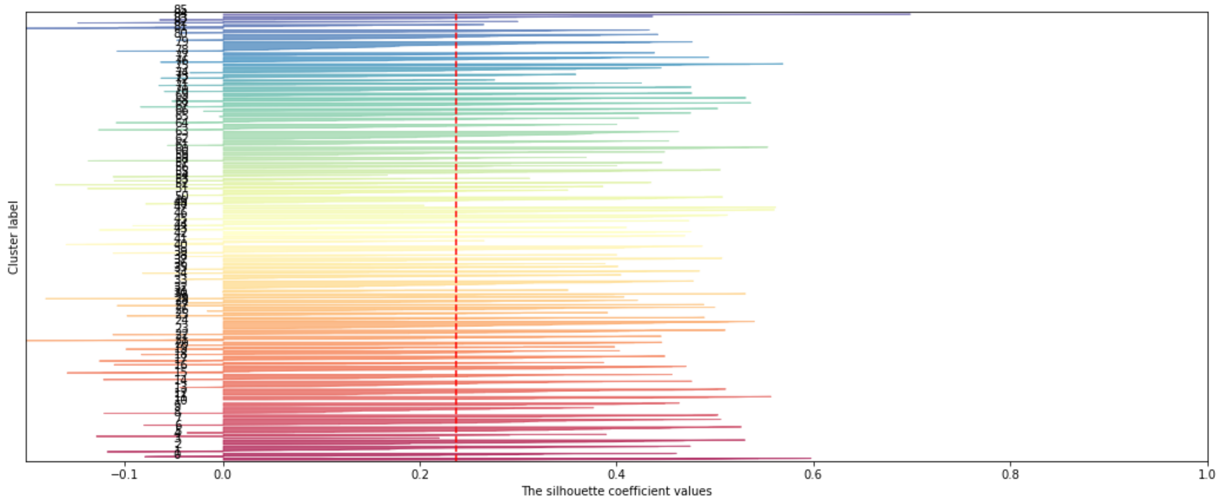
In [53]:
```
# this method find the optimal clasters number
AFP = AffinityPropagation(random_state=1).fit(scaled_data)
np.sort(AFP.labels_)
```

Out[53]:
```
array([ 0,  0,  0, ..., 85, 85, 85], dtype=int64)
```

In [54]:
```
sil_plot(AFP, scaled_data, len(set(AFP.labels_)))
```

```
For n_clusters = 86 The average silhouette_score is :0.237
```

Silhouette analysis for clustering on sample data with n_clusters = 86



In [55]:

```
# This result is pure, so the method doesn't work here
```

2.6 Agglomerative clustering method

In [57]:

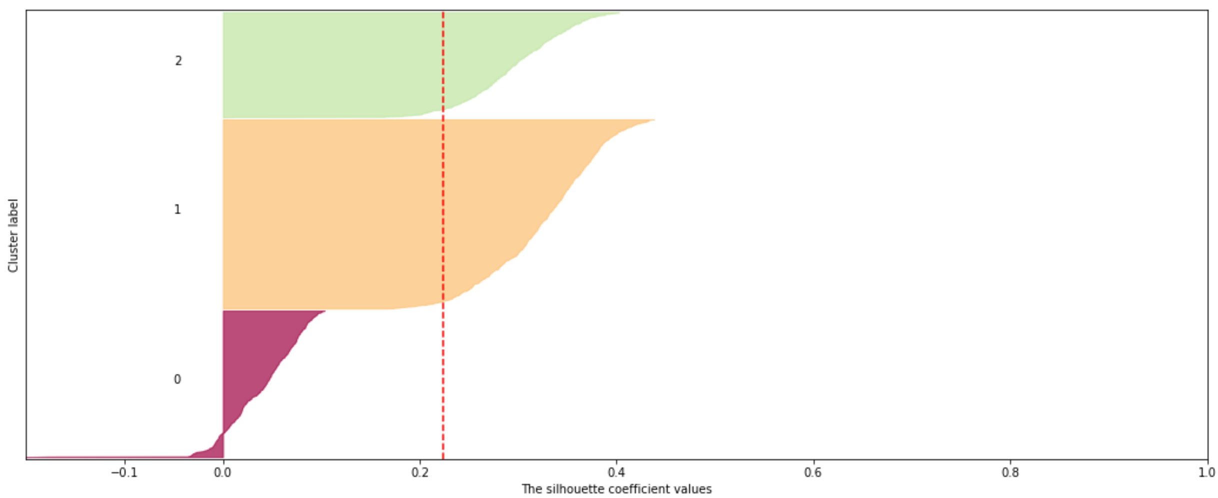
```
from sklearn.cluster import AgglomerativeClustering
```

In [58]:

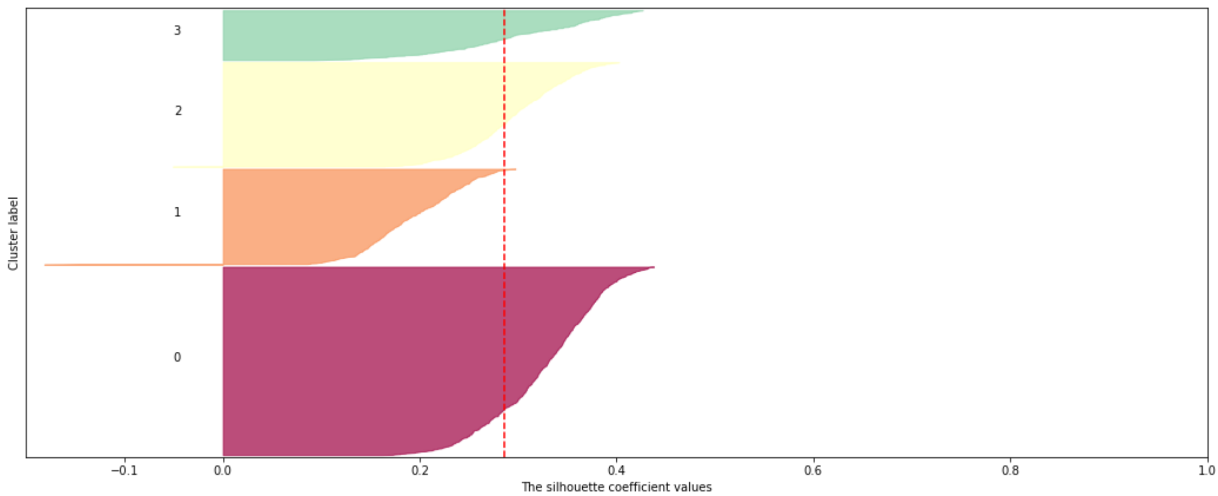
```
# try to run on # of clusters from 3 to 15, appraising the result by silhouettes
for n in range(3,16):
    AMC = AgglomerativeClustering(n_clusters=n)
    sil_plot(AMC, scaled_data, n)
```

```
For n_clusters = 3 The average silhouette_score is :0.223
For n_clusters = 4 The average silhouette_score is :0.286
For n_clusters = 5 The average silhouette_score is :0.244
For n_clusters = 6 The average silhouette_score is :0.272
For n_clusters = 7 The average silhouette_score is :0.272
For n_clusters = 8 The average silhouette_score is :0.273
For n_clusters = 9 The average silhouette_score is :0.236
For n_clusters = 10 The average silhouette_score is :0.237
For n_clusters = 11 The average silhouette_score is :0.244
For n_clusters = 12 The average silhouette_score is :0.242
For n_clusters = 13 The average silhouette_score is :0.243
For n_clusters = 14 The average silhouette_score is :0.234
For n_clusters = 15 The average silhouette_score is :0.233
```

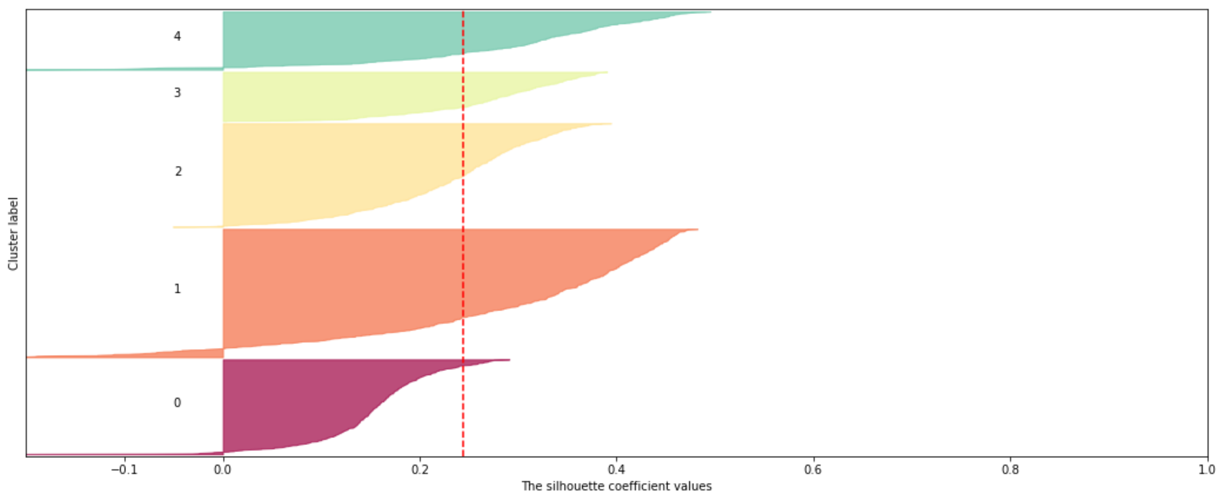
Silhouette analysis for clustering on sample data with n_clusters = 3



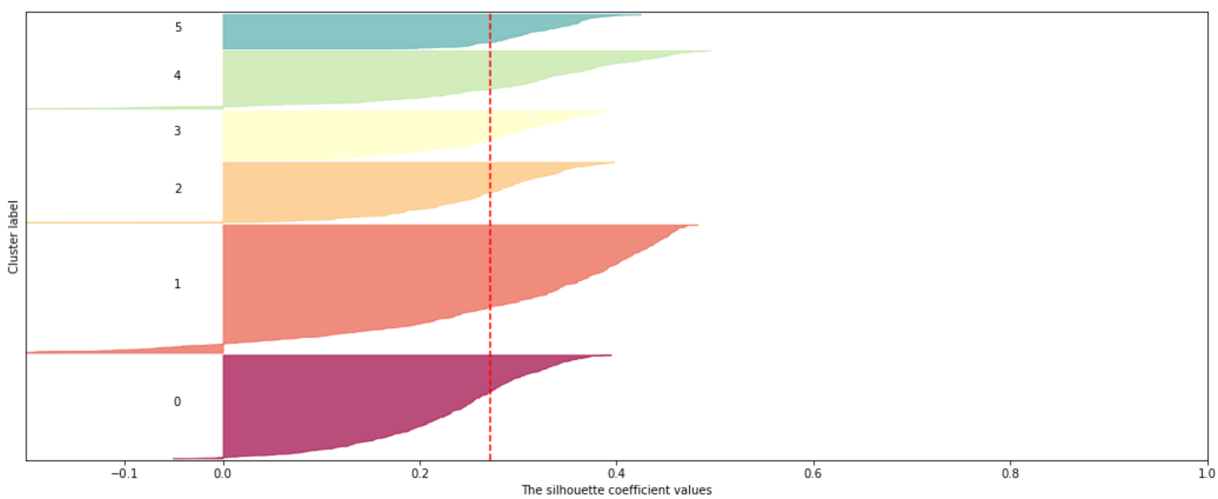
Silhouette analysis for clustering on sample data with n_clusters = 4



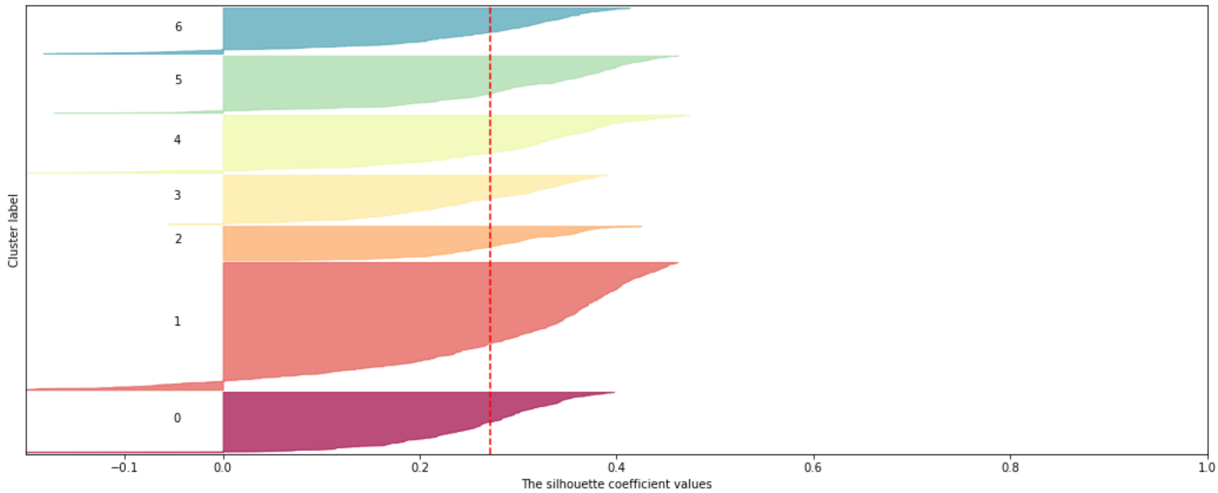
Silhouette analysis for clustering on sample data with n_clusters = 5



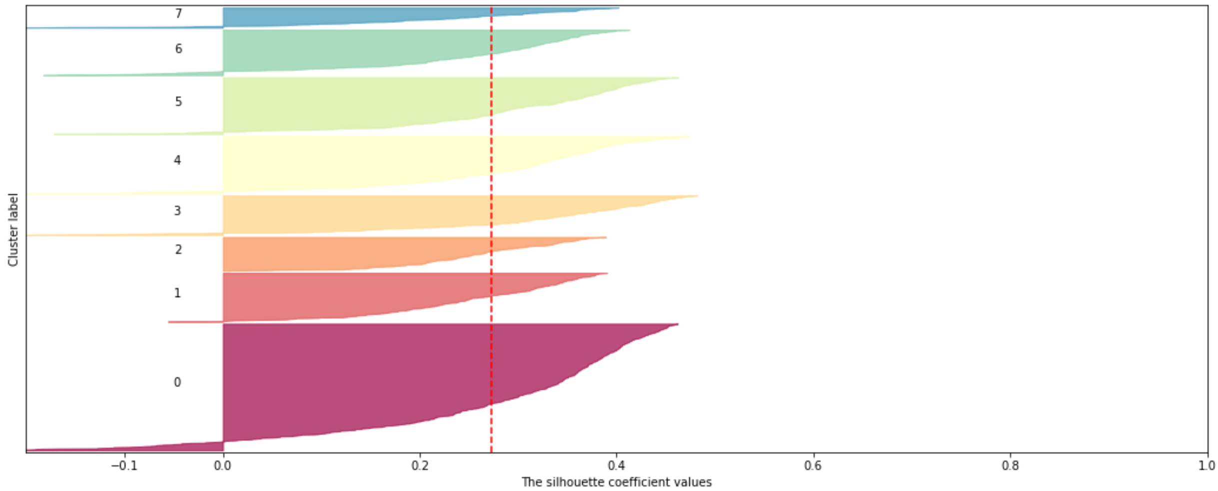
Silhouette analysis for clustering on sample data with n_clusters = 6



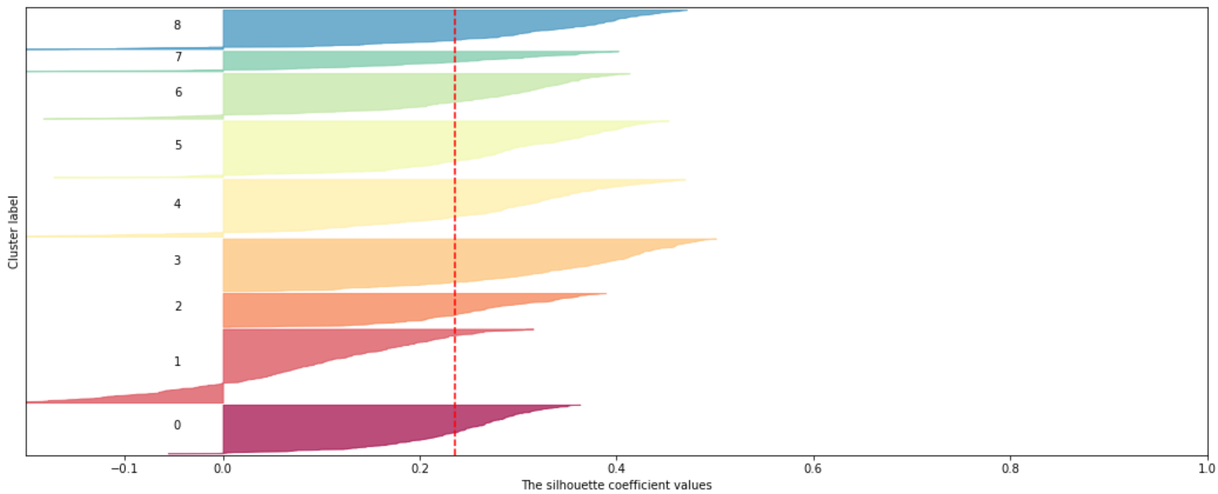
Silhouette analysis for clustering on sample data with n_clusters = 7

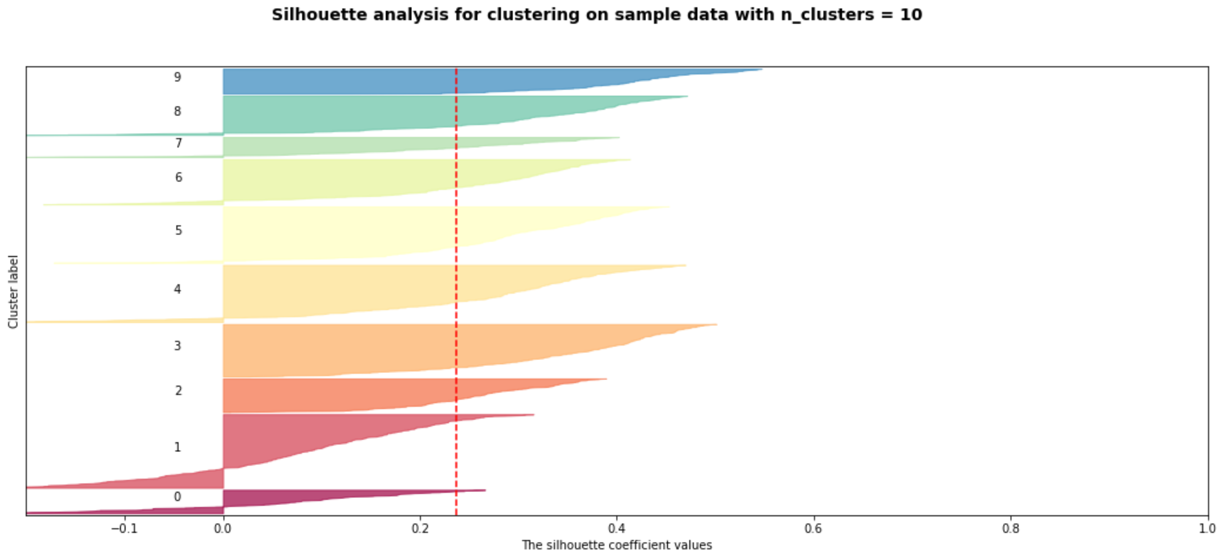


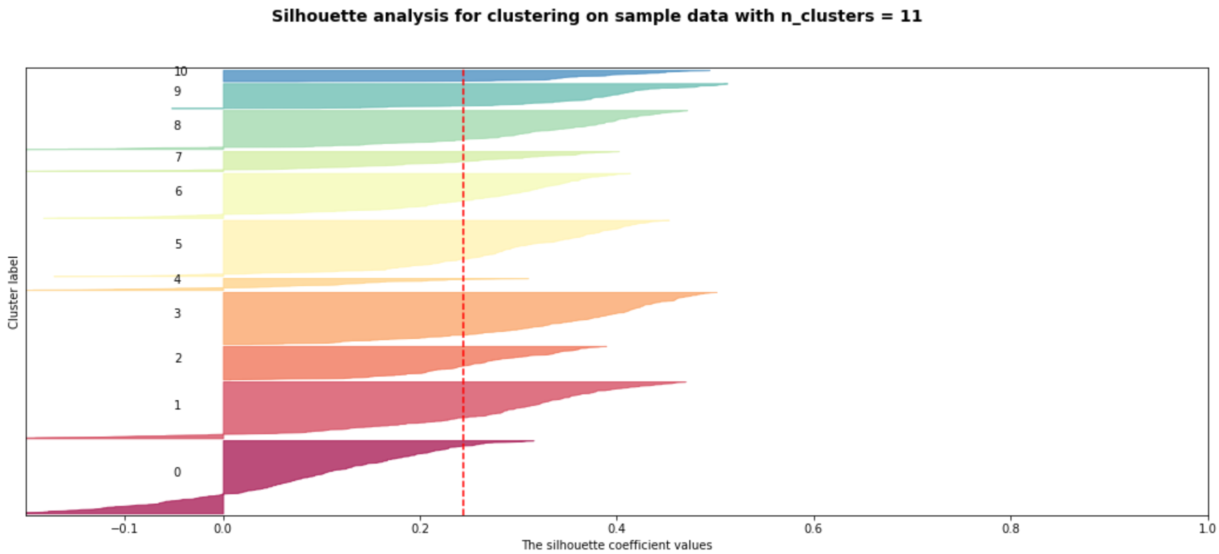
Silhouette analysis for clustering on sample data with n_clusters = 8

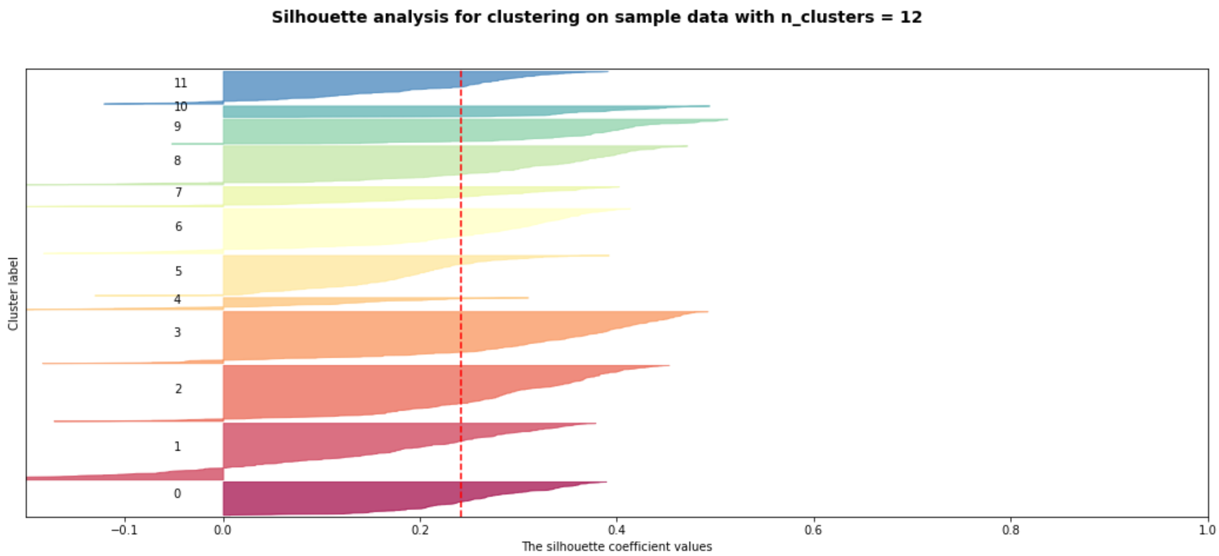


Silhouette analysis for clustering on sample data with n_clusters = 9

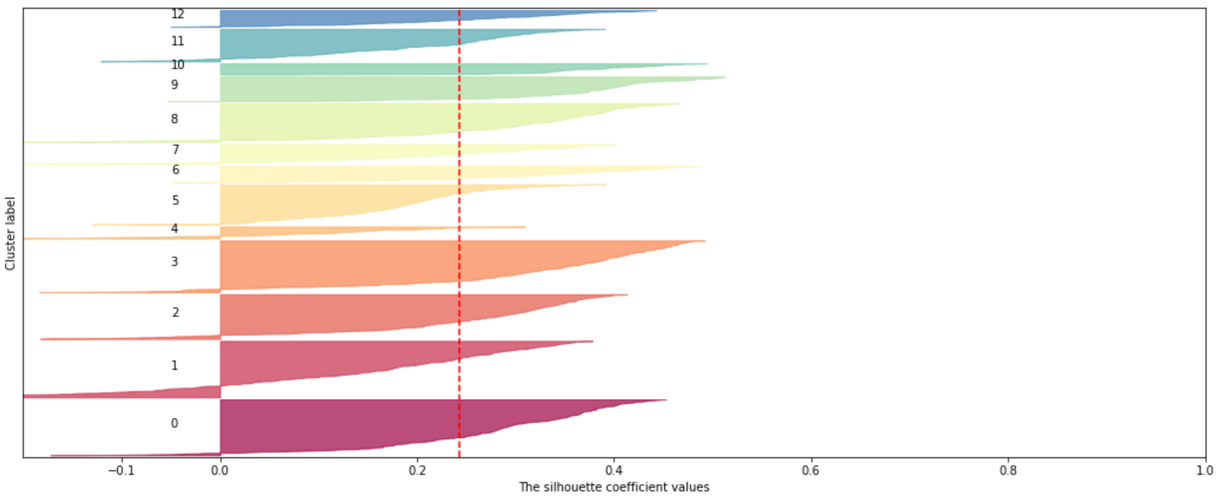



Silhouette analysis for clustering on sample data with n_clusters = 10
Cluster label
The silhouette coefficient values

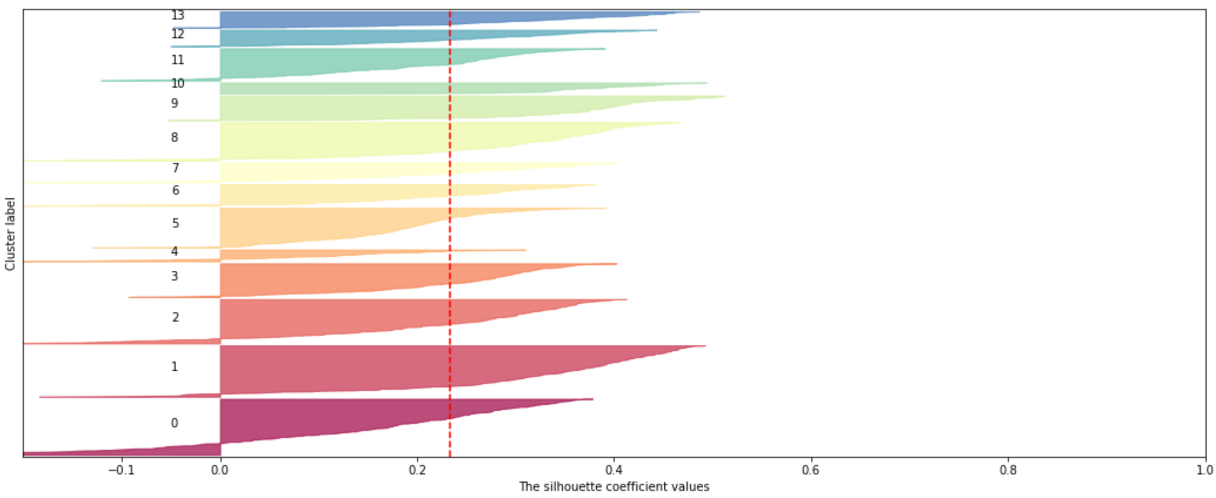

Silhouette analysis for clustering on sample data with n_clusters = 11
Cluster label
The silhouette coefficient values


Silhouette analysis for clustering on sample data with n_clusters = 12
Cluster label
The silhouette coefficient values

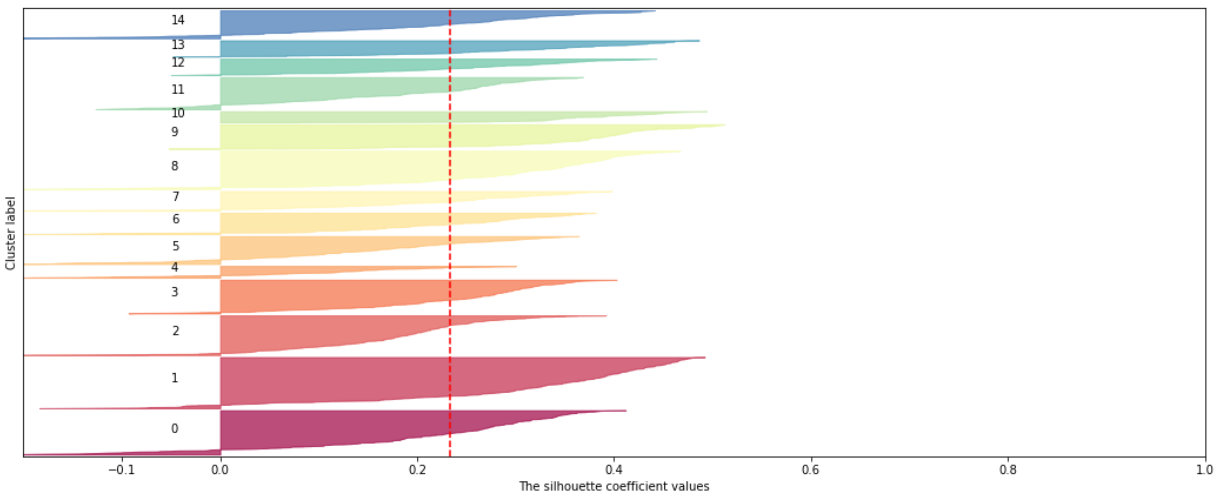
Silhouette analysis for clustering on sample data with n_clusters = 13



Silhouette analysis for clustering on sample data with n_clusters = 14



Silhouette analysis for clustering on sample data with n_clusters = 15



In [59]:

```
# Here 4-clustering looks more or less, but the result is visually worse than K-Mean
```