Evaluating Clusters

Module 1 | Chapter 3 | Notebook 6

You have already grouped the data into clusters and analyzed these clusters. You used the *elbow method* to determine the number of clusters. Now it'll be interesting to know how well our cluster analysis actually worked. Unfortunately there's no definitive answer to this question. This is where you have to assess the situation yourself. However, there are several methods that can help you do this. A widely used method is the *silhouette* analysis. We'll look at these in this lesson. Afterwards you will be able to:

- Use the *silhouette method* to evaluate clusters
- Identify typical and unusual data points in clusters

Silhouette scores

Scenario: You work for an online retailer that sells various gift items. The company is mainly aimed at business customers. Your customer base is also very diverse. In order to better address the people using your online platform, the marketing department would like to get more insight into customer behavior. The customer base should be divided into groups based on orders they have placed to date.

You have been provided with data containing all orders and cancellations in a one year period, in order to gain some initial insights. You have already prepared and saved the data in *customer_data_prepared.p*.

First of all, let's start by importing and standardizing the data. Run the following cell to do this.

In [1]:
```
# import the data from pickle
import pandas as pd
df_customers = pd.read_pickle('customer_data_prepared.p')

# import, instantiate and fit StandardScaler
from sklearn.preprocessing import StandardScaler
standardizer = StandardScaler()
standardizer.fit(df_customers)

# standardize the data
arr_customers_std = standardizer.transform(df_customers)

df_customers.head()
```

Out[1]:

CustomerID	InvoiceNo	Revenue	Quantity	RevenueMean	QuantityMean	PriceMean	DaysBetweenIn
12347.0	7	4310.00	2458	615.714286	351.142857	1.753458	
12348.0	4	1437.24	2332	359.310000	583.000000	0.616312	
12349.0	1	1457.55	630	1457.550000	630.000000	2.313571	
12353.0	1	89.00	20	89.000000	20.000000	4.450000	
12354.0	1	1079.40	530	1079.400000	530.000000	2.036604	

The rows indicate the customer numbers. The following data dictionary explains what the columns represent:

Column number	Column name	Type	Description
0	`'Revenue'`	continuous (`float`)	total revenue in GBP
1	`'Quantity'`	continuous (`int`)	total number of items purchased
2	`'InvoiceNo'`	continuous (`float`)	the order number
3	`'RevenueMean'`	continuous (`float`)	average revenue per order
4	`'QuantityMean'`	continuous (`float`)	average number of items per order
5	`'PriceMean'`	continuous (`float`)	average item price
6	`'DaysBetweenInvoices'`	continuous (`int`)	average number of days between orders
7	`'DaysSinceInvoice'`	continuous (`int`)	number of days since the customer's last order

Now we have to cluster the data again. Use the parameters `n_clusters=7` and `random_state=0`. The standardized data is stored in the variable `arr_customers_std`. Name your model `model`.

Tip: `KMeans` is located in the submodule `sklearn.cluster`

In [2]:
```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=7, random_state=0)
model.fit(arr_customers_std)
```

Out[2]:
```
KMeans(n_clusters=7, random_state=0)
```

Now we have our clusters again. So far, we have used the distance between the points in a cluster and the associated centroid to evaluate the quality of the clusters and thus determine the optimal number of clusters. Next, we'll look at *silhouette scores*. This is defined for each data point using the following formula:

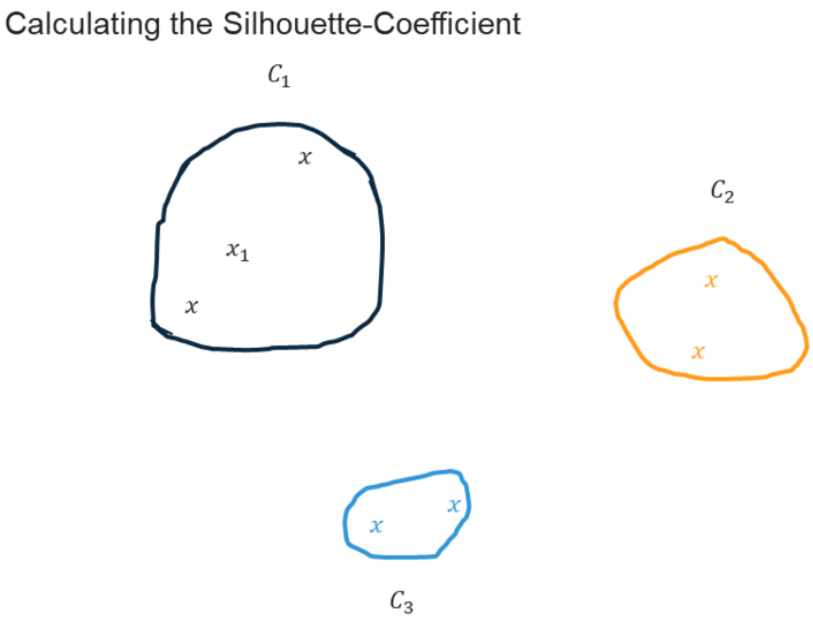
$$\mathrm{silhouette\_coef}(x_i) = \frac{\mathrm{mean\_dist_{cc}}(x_i)} - \mathrm{mean\_dist_{mc}}(x_i)}}{\mathrm{max}\left(\mathrm{mean\_dist_{cc}}(x_i)} , \mathrm{mean\_dist_{mc}}(x_i)} \right)}$$

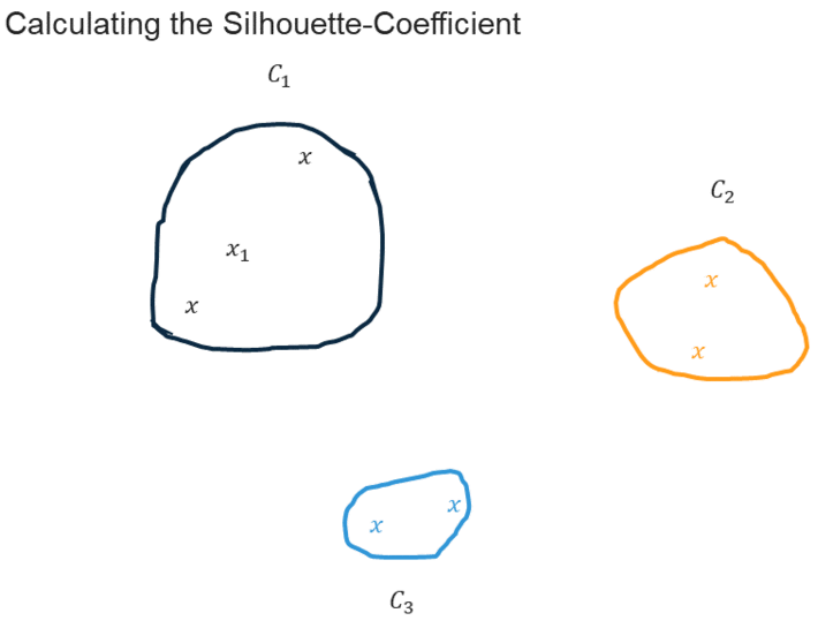
Where $\mathrm{mean\_dist_{cc}}$ is the average distance to the points of the closest cluster. $\mathrm{mean\_dist_{mc}}$ is the average distance to the points in the same cluster (*my cluster*). The maximum function in the divider only selects the maximum of the two average distances.

The score can take values in the range from -1 to 1 and can be interpreted as follows:

- If $\mathrm{silhouette\_coef}$ is positive, the point is closer to its own* The higher the value, the more typical the point is for the cluster.
- If $\mathrm{silhouette\_coef}$ is negative, the point is further away from its own cluster. It could have been assigned to the wrong cluster.
- A score of close to zero implies that the data point is located equidistant between the clusters. This may be because the clusters overlap.

Note that the position of the centroids is not directly reflected in the silhouette scores.





The following animation demonstrates how the calculations are carried out:

The average of all *silhouette scores* is the *silhouette coefficient*, and `sklearn` provides a function to calculate it. This is located in `sklearn.metrics` and is called `silhouette_score`. This function uses the `X` parameter for the data points (`arr_customers_std`) and the `labels` attribute for the cluster assignment (`model.labels_`). It uses the Euclidean distance by default. What *silhouette score* do you get for the calculated clusters?

In [3]:
```
from sklearn.metrics import silhouette_score
silhouette_score(
                arr_customers_std,
                model.labels_,
                metric='euclidean',
            )
```

Out[3]:
```
0.3245940196028819
```

The average *silhouette score* is about 0.32. It is positive, so most of the data points seem to have been allocated meaningfully. It isn't particularly high. This suggests that the clusters do not form strictly separated islands with large distances between them.

We can also calculate the silhouette score for each data point individually. We can then use this, for example, to investigate unusual data points or to examine the quality of individual clusters. The calculation carries out the `silhouette_samples` function from `sklearn.metrics` for us. It takes the same parameters as `silhouette_score`. The return value is an array with one value for each data point.

Calculate the silhouette scores for all the data points and store them in `arr_sil`. What is the minimum value you receive? Which cluster does this belong to? Which data point is it for? Print the relevant row of `df_customers`.

In [4]:
```
from sklearn.metrics import silhouette_samples
import numpy as np

arr_sil = silhouette_samples(arr_customers_std,
                             model.labels_,
                             metric='euclidean')
print(np.min(arr_sil))

mask = np.min(arr_sil) == arr_sil

print('Minimal coefficent:', arr_sil[mask])
print('In cluster:', model.labels_[mask])
df_customers.iloc[mask, :]
```

```
-0.230479885246537
Minimal coefficent: [-0.23047989]
In cluster: [6]
```

Out[4]:

CustomerID	InvoiceNo	Revenue	Quantity	RevenueMean	QuantityMean	PriceMean	DaysBetweenIn
12501.0	1	1945.68	1714	1945.68	1714.0	1.135169	

The data point with the lowest score is part of cluster 6 and belongs to customer number 12501. As a reminder: From the last notebook we know that Cluster 6 is only a small cluster with 15 customers - at least for our solution. Your results can be slightly different, but see if they match qualitatively. These people usually ordered things more frequently, and the time between orders

was less than one month. They generated a high turnover and a ordered a large quantity of items. Here are the median values from the last exercise for this cluster:

This data point stands out not only because of a lower order quantity, but also because of the high average price of their orders.

The lengths of time since the last order were longest in cluster 4 (a median of 268 days). Maybe this is why the data point fits better into this cluster? We can try this out by calculating the average distance from this point to the data points in the other clusters. We'll perform the calculation in the following code cell.

To do this, let's import `euclidean_distances`. We'll go through all the clusters with a loop. The associated values are selected each time using a Boolean mask. We'll then calculate the distance to these points. Since we have only one point as the second argument, we need to rearrange the values of the point using `arr_customers_std[mask].reshape(1, -1)`. If we don't do this, then `sklearn` will point this out explicitly in an error message. At the end we'll print the mean distance to the points in the respective cluster.

Run the following cell:

In [8]:
```
from sklearn.metrics import euclidean_distances

mask = np.min(arr_sil) == arr_sil  # create a mask to select the data point with the w
for cluster in range(7):  # iterate through every cluster
    arr_cluster = arr_customers_std[model.labels_ == cluster]  # select the values of
    distances = euclidean_distances(arr_cluster, arr_customers_std[mask].reshape(1, -1
    print(cluster,':', np.mean(distances))  # calculate and print the mean
```

```
0 : 6.572108599885569
1 : 7.023203640687759
2 : 35.52384001252933
3 : 56.3159201170194
4 : 6.201419017770749
5 : 15.938032039031867
6 : 7.521559179914407
```

In fact, the distance from the point with the minimum silhouette score to the points in cluster 4 is the smallest. You could therefore assume that it's a better fit in this cluster. So why was it assigned to cluster 6? This is because the distance to the cluster center is smaller. The cluster centers are based on mean values and mean values are not particularly stable when faced with outliers. This point therefore pulls the cluster center towards itself a bit, while the other cluster centers are located somewhat further away.

Run the following code cell to calculate the distances to the cluster centers:

In [9]:
```
for cluster in range(7):  # iterate through every cluster
    distances = euclidean_distances(model.cluster_centers_[cluster].reshape(1, -1), ar
    print(cluster,':', np.mean(distances))
```

```
0 : 6.452036887654675
1 : 6.951986776891534
2 : 34.14241357901757
3 : 56.3159201170194
4 : 6.123261447409446
5 : 14.896878630001986
6 : 5.952491560841114
```

As expected, Cluster Centre 6 is the closest. But Cluster Centre 4 is not much further away.

Congratulations: You have calculated the silhouette scores for the data points and used these to identify a data point that doesn't really fit into any of the groups. This is not unusual when you are using real data. Now let's look at how we can use the silhouette method to visually assess the state of our clusters.

Visualizing the silhouette scores

A visualization of the silhouette scores helps to estimate the quality of the clusters. A bar chart is suitable for this purpose, in which the values of the scores are shown on the x-axis and the number of the data points on the y-axis. First import the `matplotlib.pyplot` module with its conventional alias.

In [10]:
```
import matplotlib.pyplot as plt
```

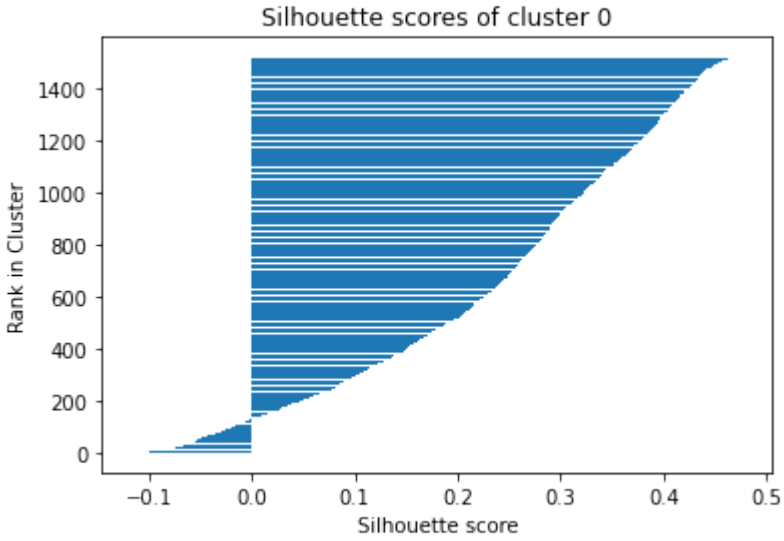
We will now create a larger visualization, which we will add to step by step. The first step is to represent a single cluster.

Create a Boolean mask for `arr_sil`, to select only the values belonging to cluster 0. Store the number of values belonging to cluster 0 as `sv_len`. Sort the silhouette scores of cluster 0 by size and save them as `sv_sorted`.

Then create a *figure* and an *axes*. Store these as `fig` and `ax`. Then use `ax.barh()`. First pass `range(sv_len)` to this method for the vertical axis values and then `sv_sorted` for the horizontal axis values.

Tip: This visualization is not about finding the position of data points with a specific value. The sorting of the y-axis can therefore be different to the x-axis.

In [32]:
```
mask = model.labels_ == 0
sv_len = len(arr_sil[mask])

sv_sorted = np.sort(arr_sil[mask])

fig, ax = plt.subplots()
ax.barh(range(sv_len), sv_sorted)
plt.title("Silhouette scores of cluster {}".format(str(0)))
plt.ylabel("Rank in Cluster")
plt.xlabel("Silhouette score");
```



After a bit of tweaking, the visualization of the scores for cluster 0 looks something like this:

In this figure you can see that a large number of our data points are included in cluster 0, because it depicts over 1400 scores (with fewer than 3000 data points in total). The cluster has some data points that are closer to the points in a neighboring cluster, these have a negative score.

The next step is to display all the clusters. Do this with a loop that iterates through each cluster and displays it. Use the code from just now as a guide. Define the *figure* and *axes* before the loop. All clusters should be plotted on the same *axes*.

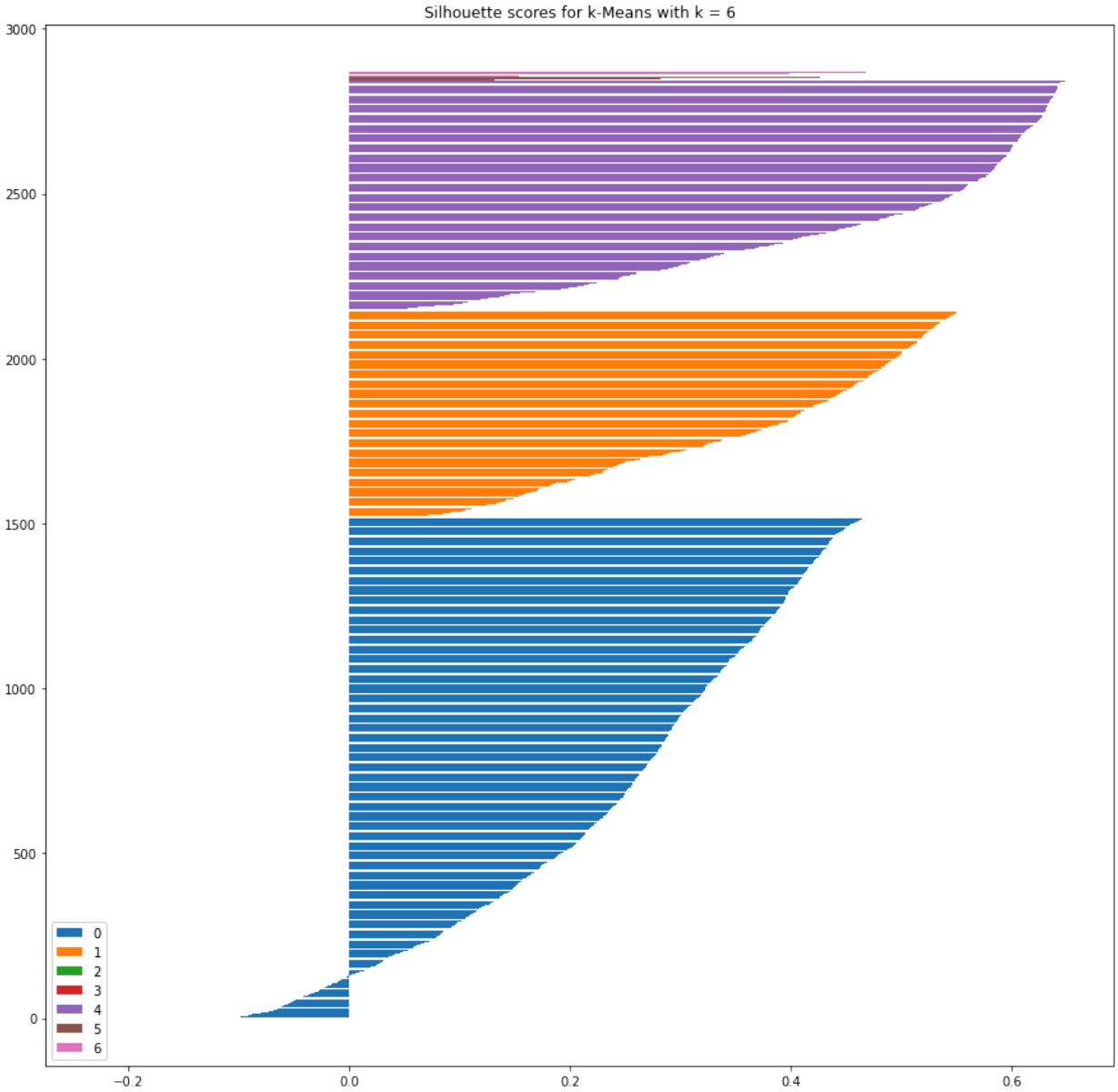
Tips:

- pass the parameter `figsize=(15, 15)` when you create the figure to make the visualization bigger. Otherwise you will not see all the details in the graph.
- The number of points in a cluster can vary. The scores of different clusters are to be displayed on top of each other. So define the variables `start` and `end` before the loop. Increase `end` within the loop by the number of values in the current cluster. Always set `start` to `end` after plotting the values. You can use these two variables in `range()` when plotting. This allows you to stack the clusters on top of each other in the visualization.
- pass the parameter `label=c` to `ax.barh()`. Call `ax.legend()` after the end of the loop. As a result, the number representing c is used as the legend entry in the bar chart legend for this part of the data.

In [42]:
```
fig, ax = plt.subplots(figsize=(15, 15))
start = 0
end = 0

for cluster in range(7):
    mask = model.labels_ == cluster

    sv_len = len(arr_sil[mask])
    sv_sorted = np.sort(arr_sil[mask])
```

```
    end = end + sv_len
    ax.barh(range(start, end), width=sv_sorted, label = cluster)
    start = end
ax.legend()
ax.set_title("Silhouette scores for k-Means with k = {}".format(cluster))
```

Out[42]:
```
Text(0.5, 1.0, 'Silhouette scores for k-Means with k = 6')
```



We can see that the data consists mainly of three large clusters. So the other four clusters only contain a few customers, so we can hardly see them in the diagram.

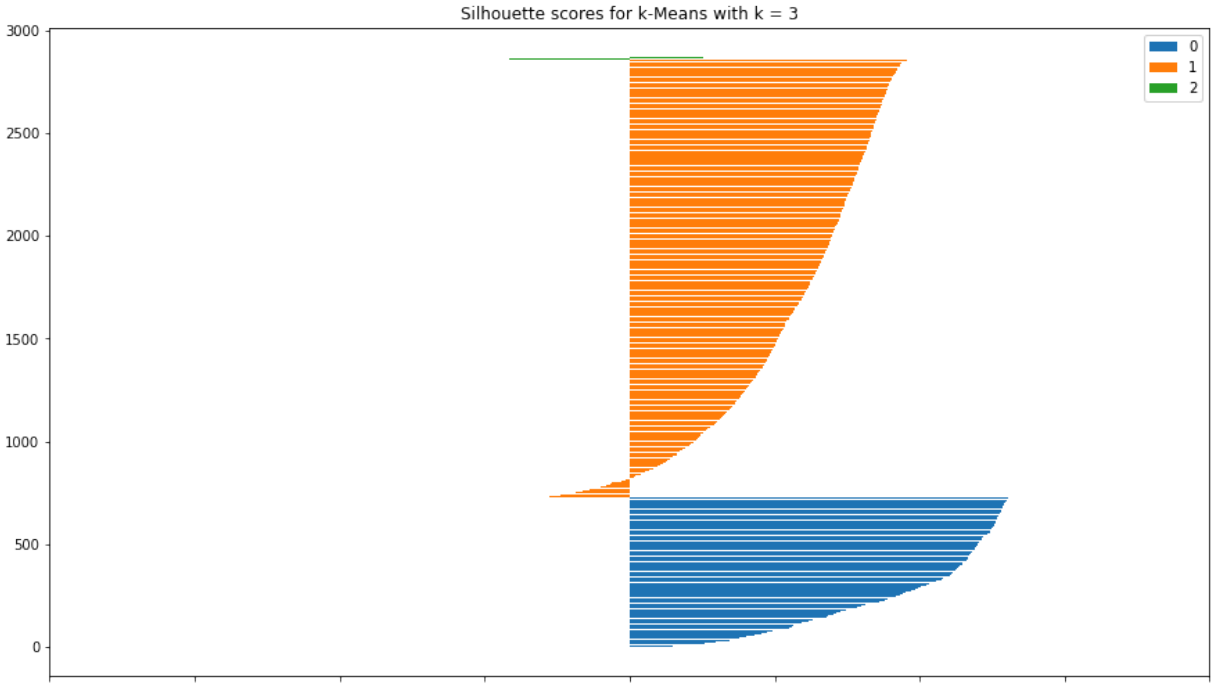
Clusters 1, 3 and 4 don't have any negative scores. This means that their data points form a relatively compact cluster. The other clusters exhibit points with negative values which is not necessarily visible in the plot as the negative values are small and difficult to depict. Negative silhouette scores mean that these points are closer to the points of another cluster. This blurs the boundaries between the clusters.

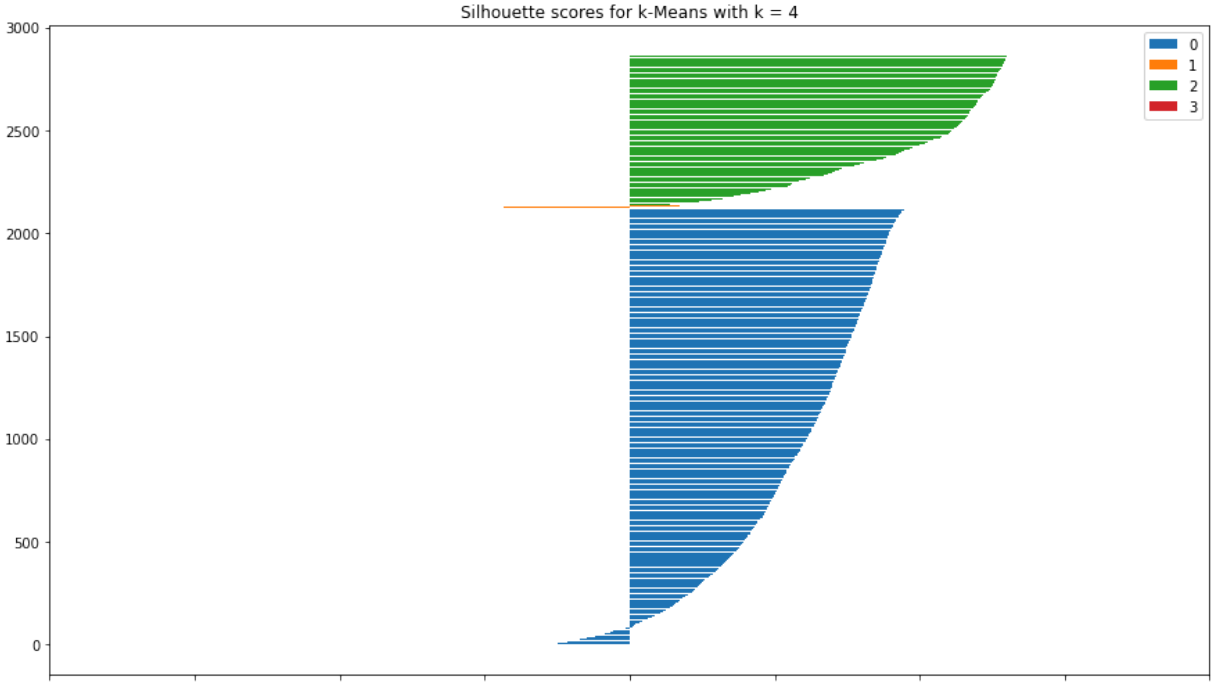
Cluster 3 doesn't appear because it only has one data point and therefore a scores of 0. So this is a data point that doesn't really fit anywhere.

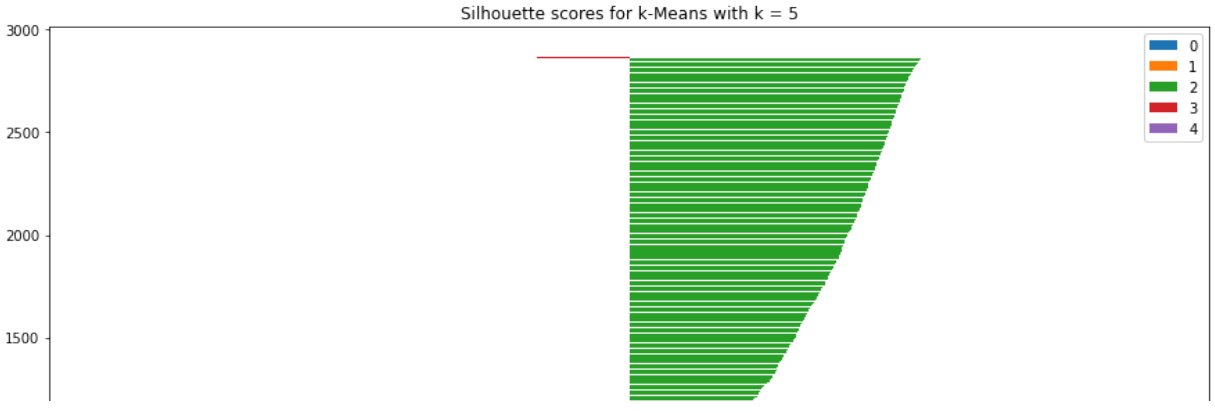
The final step is to create this loop for different numbers of cluster centers. So your previous code is inside another loop where the model is newly instantiated and fitted with different `n_clusters` values. The silhouette scores are also calculated again. This is already prepared in the following code cell. All you have to do is run it. This can take a few seconds.

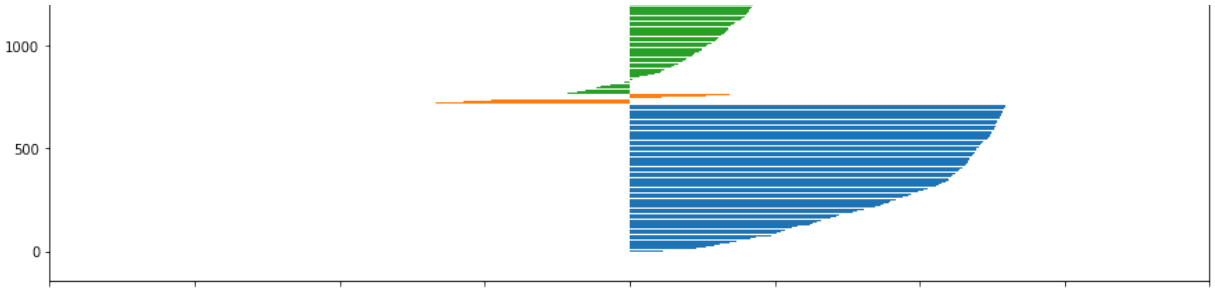
In [41]:
```
fig, axs = plt.subplots(nrows = 5,
                        figsize=(15, 50),
                        sharex=True)  # create figure with 5 axes because we visualize

for n_cluster in range(3,8):
    # fit new KMeans with n_clusters
    model = KMeans(n_clusters=n_cluster, random_state=0)
    model.fit(arr_customers_std)
    #calculate silhouette coefficient and scores
    print('Number of clusters:',n_cluster,'overall score:', silhouette_score(arr_custc
    arr_sil = silhouette_samples(arr_customers_std, model.labels_)

    #plot the scores
    start = 0  # start and end are needed to plot one group above the other
    end = 0
    for cluster in range(n_cluster):
        mask = model.labels_ == cluster  # create a mask to select data points from th
        sv_len = len(arr_sil[mask])   # the length is needed to increase end for plott
        sv_sorted = np.sort(arr_sil[mask])  # sort the silhouette scores within each c
        end = end + sv_len  # increase end: be able to get a range with the length of
        axs[n_cluster-3].barh(range(start, end), sv_sorted, label=cluster)  # plot the
        start = end  # increase start: the next cluster will be plotted above this one
    axs[n_cluster-3].set_title("Silhouette scores for k-Means with k = {}".format(n_cl
    axs[n_cluster-3].legend()
    plt.xlim([-1,1])
```

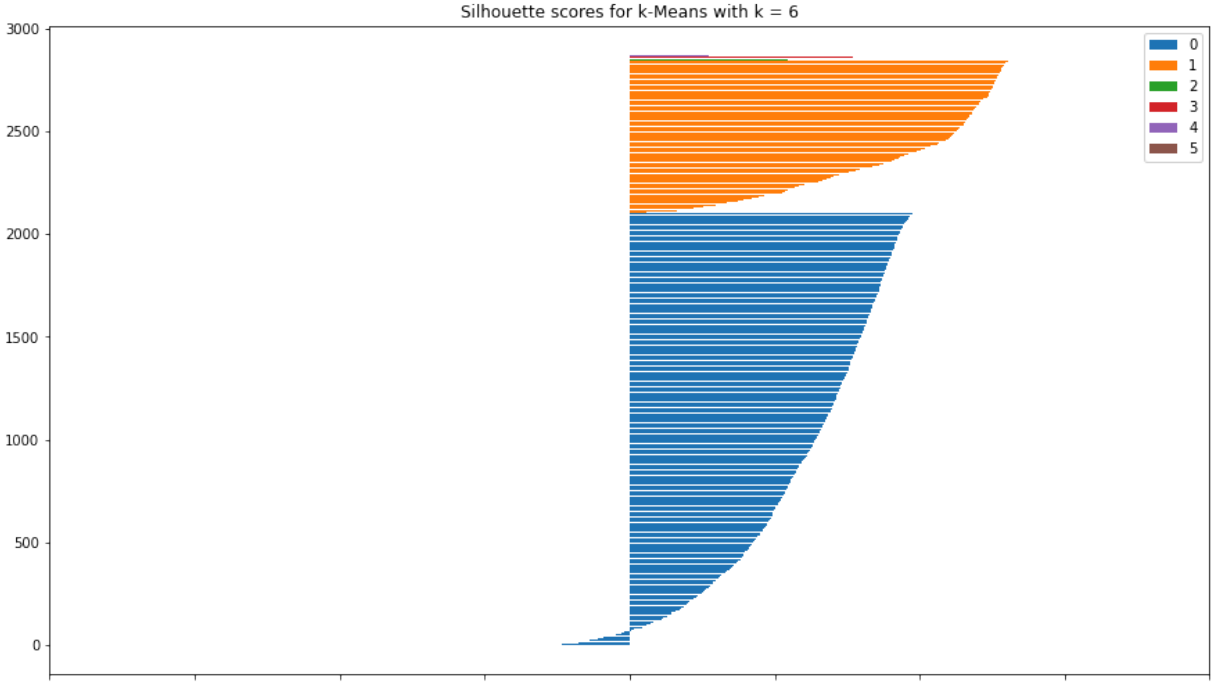
```
Number of clusters: 3 overall score: 0.3336723147801023
Number of clusters: 4 overall score: 0.33492380373780956
Number of clusters: 5 overall score: 0.3519451101310087
Number of clusters: 6 overall score: 0.3459763772746656
Number of clusters: 7 overall score: 0.3245940196028819
```

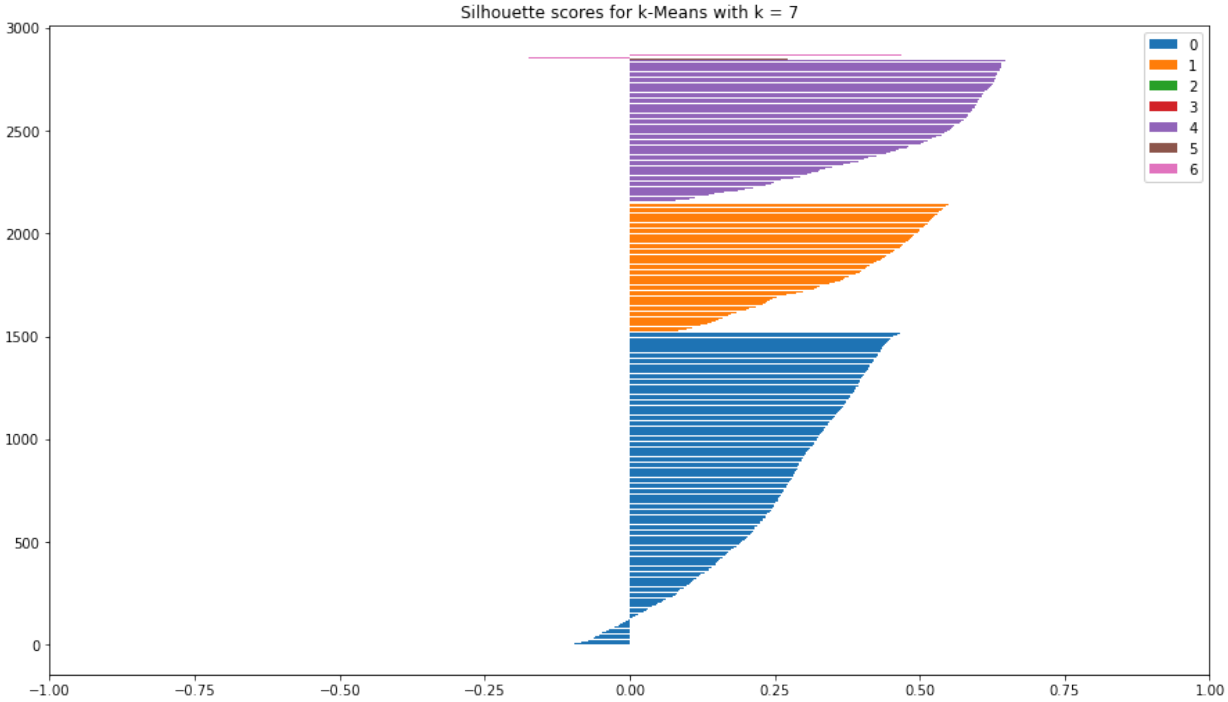

Silhouette scores for k-Means with k = 3
3000
2500
2000
1500
1000
500
0
0
1
2


Silhouette scores for k-Means with k = 4
3000
2500
2000
1500
1000
500
0
0
1
2
3


Silhouette scores for k-Means with k = 5
3000
2500
2000
1500
0
1
2
3
4


1000
500
0


Silhouette scores for k-Means with k = 6
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5


Silhouette scores for k-Means with k = 7
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

The naming of the individual clusters may differ for each number of clusters, as this is determined by chance. In this case, their arrangement is determined by the names.

Nevertheless, we can see that our original Cluster 0 always makes up a large part of the data, even if we use fewer clusters. Only once we have 7 clusters is a large part of the data from this cluster reassigned to a new cluster (cluster 1 in our case with `n_clusters = 7`). Cluster 0 and 1 are therefore relatively similar. Sometimes points with extreme values are already assigned as a separate cluster with as few clusters as `n_clusters = 3`. The structure of the scores in this data does not differ much for different cluster numbers. The number is therefore up to you. Depending on how small the customer groups should be, we can define more or fewer clusters.

Now you know two numerical values to describe the quality of our clusters: The average *within-cluster sum of squares* and the average silhouette score, known as the *silhouette coefficient*.

With the *within-cluster sum of squares*, only the distance of each data point to its own cluster center is taken into account. The distance to other clusters is ignored. It is not standardized. So if the clusters contain more data points, this value also increases. Therefore the value of the *within-cluster sum of squares* is not suitable for comparing different data sets. We should only use it to determine the number of clusters with the *elbow method*.

With the silhouette method, the distance to the cluster center is not directly included in the calculation. Instead, this metric focuses on the distance between the data points of one cluster and the nearest cluster. If the distances between the clusters increase, the average *silhouette coefficient* increases. It is also standardized to a value between -1 and 1. This value can be used to compare the structure of different data sets. The higher the average *silhouette coefficient*, the more isolated the clusters making up the data point are. Furthermore, you can use the silhouette scores for the individual data points to check whether a specific point has been well assigned.

`sklearn` also offers other measures to evaluate clusters. If you are interested, then have a look here.

Congratulations: You have learned another way, the silhouette method to estimate the structure of clusters visually and to validate the number of clusters. You also get a feeling for how well the clusters are separated from each other. Your analysis has shown that there are a great deal of customers whose behavior clearly stands out from the others. Therefore, in the next lesson, you will try another clustering algorithm that determines the number of clusters itself. Perhaps this is even better suited to identifying customers who stand out clearly from the rest.

Remember:

- The silhouette coefficient characterizes data points using the distance between points in one cluster and the nearest cluster
- Negative scores show that a data point has possibly been incorrectly assigned
- If there are large, clear differences, these have a high positive score

- You can use the silhouette method to estimate the number of clusters

---

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.

The data was published here first: Daqing Chen, Sai Liang Sain, and Kun Guo, Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining, Journal of Database Marketing and Customer Strategy Management, Vol. 19, No. 3, pp. 197-208, 2012