- Download `RAW_recipes.csv` and put it under the `data` directory in the homework folder.
- Run the code below. The dataset is quite large, and in this assignment, for speed, you will work with a sample of the dataset. The function `get_recipes_sample` below carries out some preliminary preprocessing and returns a sample of the recipes with most frequent tags.

> *Note: Depending upon the capacity of your computer, feel free to increase or decrease the size of this sample by changing the value for `n_tags`. If you decide to go with a different value of `n_tags`, state it clearly in Exercise 2.1 so that the grader knows about it.*

In [25]:
```
orig_recipes_df = pd.read_csv("data/RAW_recipes.csv")
orig_recipes_df.shape
```

Out[25]: (231637, 12)

In [26]:
```
def get_recipes_sample(orig_recipes_df, n_tags=300, min_len=5):
    orig_recipes_df = orig_recipes_df.dropna()  # Remove rows with NaNs.
    orig_recipes_df = orig_recipes_df.drop_duplicates(
        "name"
    )  # Remove rows with duplicate names.
    # Remove rows where recipe names are too short (< 5 characters).
    orig_recipes_df = orig_recipes_df[orig_recipes_df["name"].apply(len) >=
    # Only consider the rows where tags are one of the most frequent n tags.
    first_n = orig_recipes_df["tags"].value_counts()[0:n_tags].index.tolist(
    recipes_df = orig_recipes_df[orig_recipes_df["tags"].isin(first_n)]
    return recipes_df
```

In [27]:
```
recipes_df = get_recipes_sample(orig_recipes_df)
recipes_df.shape
```

Out[27]: (9100, 12)

In [28]:
```
recipes_df["name"]
```

Out[28]:
```
42        i yam what i yam  muffins
101       to your health  muffins
129       250 00 chocolate chip cookies
138       lplermagronen
163       california roll   salad
                    ...
231430    zucchini wheat germ cookies
231514    zucchini blueberry bread
231547    zucchini salsa burgers
231596    zuppa toscana
231629    zydeco salad
Name: name, Length: 9100, dtype: object
```

In the rest of the homework, we will use `recipes_df` above, which is a subset of the original dataset.

2.1 Longest and shorter recipe names

rubric={points:2}

Your tasks:

1. Print the shortest and longest recipe names (length in terms of number of characters) from `recipes_df`. If there is more than one recipe with the same shortest/longest length, store **one** of them in `shortest_recipe` and/or `longest_recipe` as a **string**.

Solution_2.1

Points: 2

In [29]:
```
shortest_recipe = recipes_df.loc[recipes_df['name'].str.len().idxmin(), 'nam
longest_recipe = recipes_df.loc[recipes_df['name'].str.len().idxmax(), 'name

shortest_recipe, longest_recipe
```

Out[29]:
```
('bread', 'baked tomatoes with a parmesan cheese crust and balsamic drizzl
e')
```

2.2 More EDA

rubric={points:2}

Your tasks:

1. Create a word cloud for the recipe names. You can use the `wordcloud` package for this, which you will have to install in the course environment.

```
> conda activate cpsc330
> conda install -c conda-forge wordcloud
```

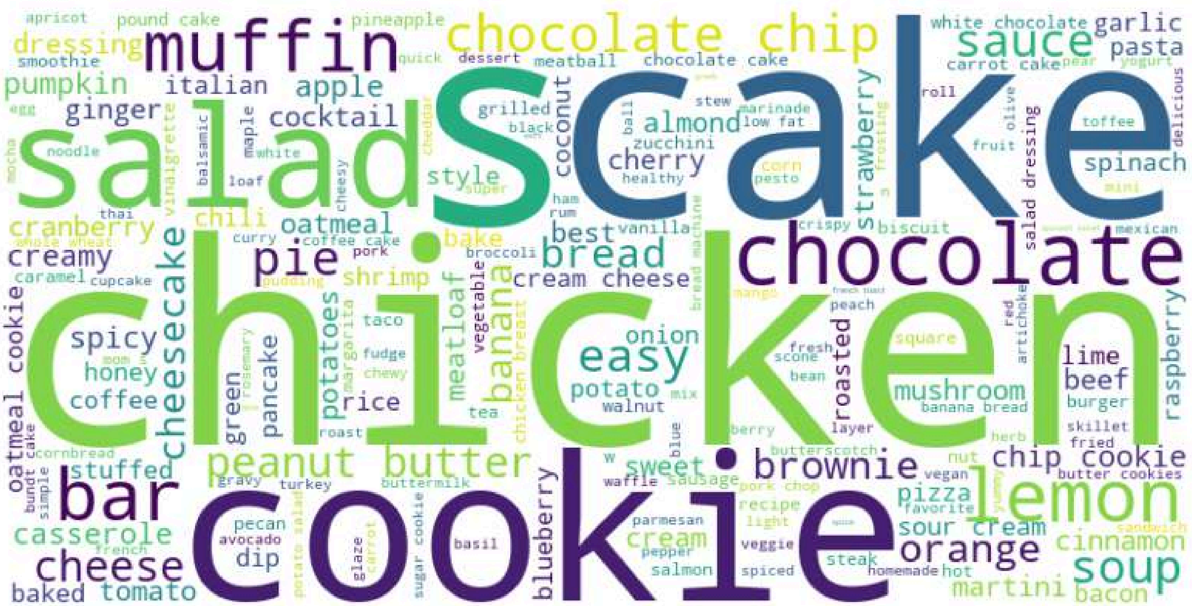
Solution_2.2

Points: 2

In [30]:
```
from wordcloud import WordCloud

recipe_names_text = " ".join(recipes_df["name"].values)
wordcloud = WordCloud(width=800, height=400, background_color='white').gener

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



2.3 Representing recipe names

rubric={points:3}

The next step is creating a representation of recipe names.

Your tasks:

1. Similar to Exercise 1, create sentence embedding representation of recipe names (`name` column in `recipes_df`). For the rest of the homework, we'll stick to the sentence embedding representation of recipe names.

   > You might have to convert the recipe names to a list (`recipes_df["name"].tolist()`) for the embedder to work *If you*

> *create a dataframe with sentence embedding representation, set the index to `recipes_df.index` so that the indices match with the indices of the sample we are working with.*
> **This might take a while to run.**

Solution_2.3

Points: 3

In [31]:
```
from sentence_transformers import SentenceTransformer

recipe_names = recipes_df["name"].tolist()
recipe_embeddings = embedder.encode(recipe_names)

recipe_emb_df = pd.DataFrame(
    recipe_embeddings,
    index=recipes_df.index
)
```

In [32]:
```
recipe_emb_df
```

Out[32]:

	0	1	2	3	4	5	6
42	-0.333475	0.227864	-0.307339	0.410549	0.917104	-0.345507	0.30581
101	-0.024523	0.246223	-0.055709	0.358273	0.454786	-0.088055	0.26036
129	-0.026562	0.194671	0.038101	-0.099181	0.653784	-0.230869	0.06451
138	-0.168002	-0.219219	0.330761	0.302196	-0.173169	0.204557	0.19239
163	0.061075	-0.333799	0.242906	0.395978	-0.466468	0.496505	-0.13675
...	...	...	...	...	...	...	..
231430	-0.009714	0.200162	0.018329	0.237817	0.748989	0.121918	0.08791
231514	-0.106532	-0.034574	0.160070	0.258504	0.882480	0.091899	0.00881
231547	0.016149	-0.053036	-0.035098	-0.021835	0.735161	0.245519	-0.15183
231596	0.026659	0.202482	0.344634	-0.138709	0.514424	0.630947	-0.00316
231629	-0.031947	-0.258190	-0.079766	-0.507932	-0.155581	0.635225	-0.12739

9100 rows × 768 columns

Exercise 3: Clustering recipe names

In this exercise you'll cluster recipe names with some of the clustering algorithms we have seen in class. This will also involve making some attempts to pick reasonable hyperparameter values for each clustering method based on the quality of the resulting clusters. For example, for KMeans, you need to specify the number of clusters in advance, which is often challenging on real-world datasets. For DBSCAN, you need to pick appropriate `eps` and `min_samples`. For hierarchical clustering, you need to pick a suitable linkage criterion, distance metric, and prune the tree so that it's possible to visualize and interpret it.

Here are some methods which may help you with picking reasonable values for the hyperparameters.

- Visualize the Elbow plot (KMeans).
- Visualize Silhouette plots.
- Visualize resulting clusters using `plot_umap_clusters` function from Exercise 1.
- Sample some recipes from each cluster, manually inspect whether there are coherent semantic themes. (For this, you may use the function `print_clusters` given below.)

> You may use the `yellowbrick` package for visualizing the Elbow plot and the Silhouette plots. You can intall it with

`conda install -c districtdatalabs yellowbrick`

Note that the process of picking reasonable hyperparameter values will be exploratory, iterative, and will involve manual inspection and judgment, as there is no ground truth to verify how well the model is doing. In your solutions, please do not include everything you try. Only present the results of the most informative trials. Add a narrative to your answer so that it's easy for the grader to follow your choices and reasoning.

In [33]:
```
def print_clusters(recipes_df, cluster_labels, n_recipes=10, replace=False,
    """
    Given recipes_df containing recipe names and cluster assignment (labels)
    sample and print n_recipes recipes per cluster.

    Parameters
    -----------
    recipe_df : pandas dataframe
        recipes dataframe containing recipe names in the "name" column
    cluster_labels : ndarray or a list
        cluster labels for each row in recipes_df
    n_recipes : int
        number of examples to sample from each cluster
```

```
    replace: bool
        replace flag to pass to the sampling of recipe names

    Returns
    -----------
    None
    """

    grouped = (
        pd.DataFrame(
            {
                "name": recipes_df["name"],
                "cluster_label": cluster_labels,
            }
        )
        .sort_values("cluster_label")
        .groupby("cluster_label")
    )

    for name, group in grouped:
        print(f"Cluster {name}")
        print(("----------").format(""))
        print("\n".join(group.sample(n_recipes, random_state=random_state)['
        print("\n\n")
```

3.1 K-Means

rubric={points:6}

Your tasks:

1. Cluster recipe titles using KMeans. Make some attempts to determine the optimal number of clusters.
2. Pick one or two best models and justify your choice.

Solution_3.1

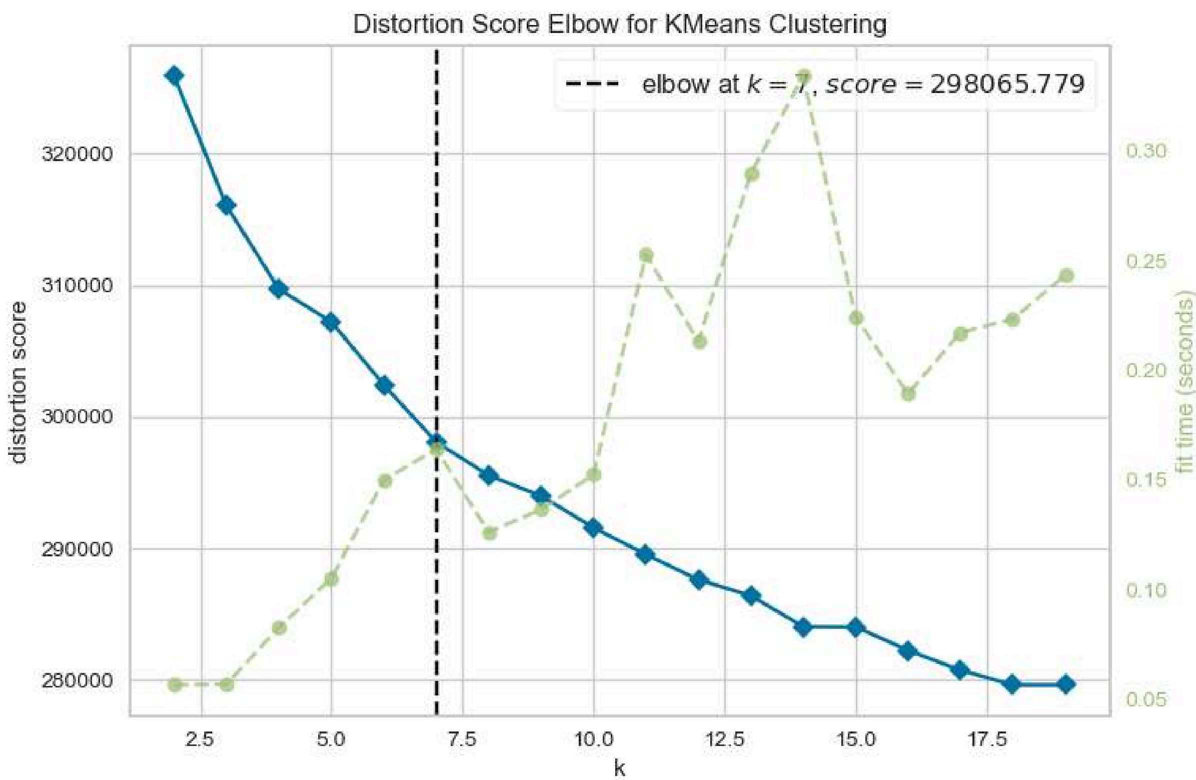
Points: 6

- The yellowbrick model gives us the optimal number of clusters as 7. However, from the plot, we can see that there's no clear elbow. So we think it's reasonable to choose K from 6 to 10.
- From the Silhouette Plots, we found with K larger than 6, the model tends to preform better than smaller K. Because the silhouette coefficient values are higher and the dropoffs are slower. Again, if we need to pick the best model, we think the model with K = 7 is the best. But there's, again, not too much difference with K =8,9, or 10.

- From Umap, we can observe similar pattern of K, where difference is not obvious. But when k = 7, the clusters are clear and reasonable.

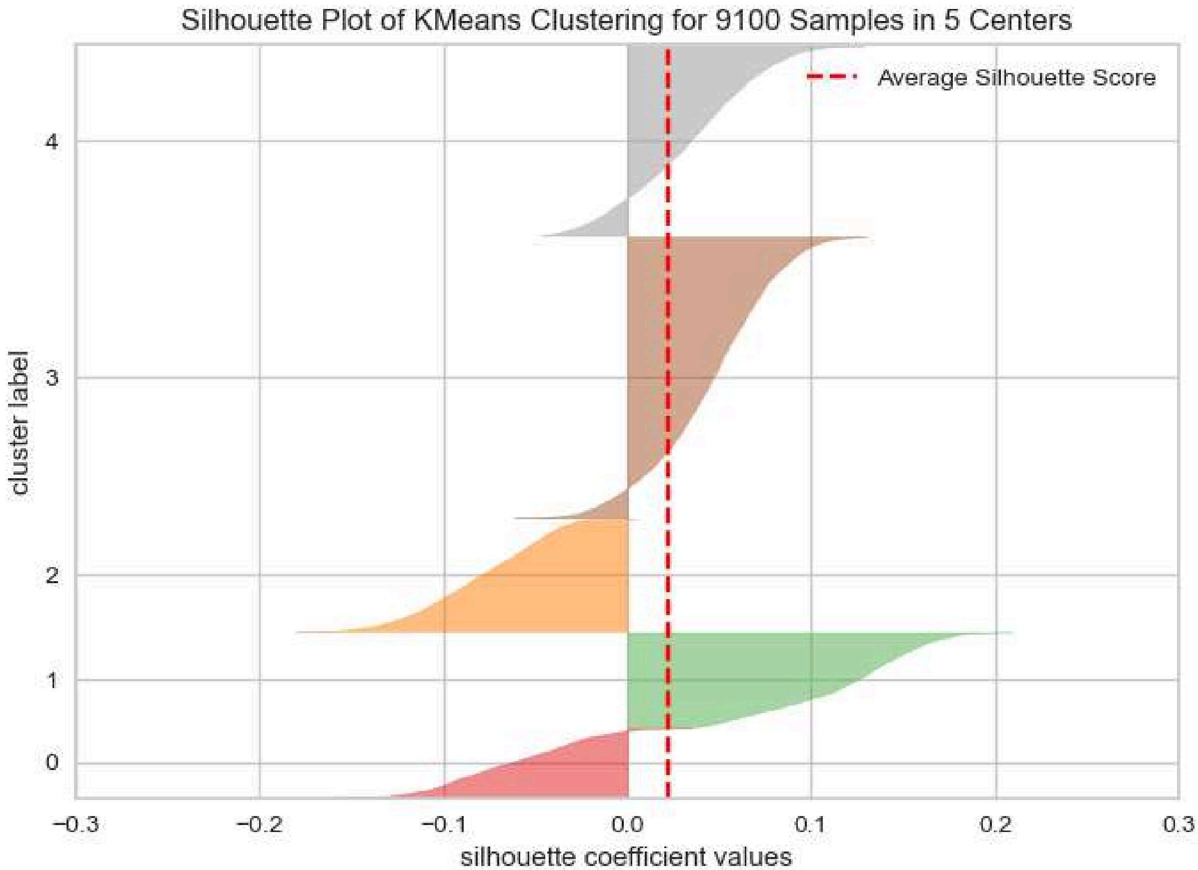
In [34]:
```
import yellowbrick
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from sklearn.metrics import silhouette_score

model = KMeans(random_state=42)
visualizer = KElbowVisualizer(
    model, k=(2,20)
)
visualizer.fit(recipe_embeddings)
visualizer.show()
```



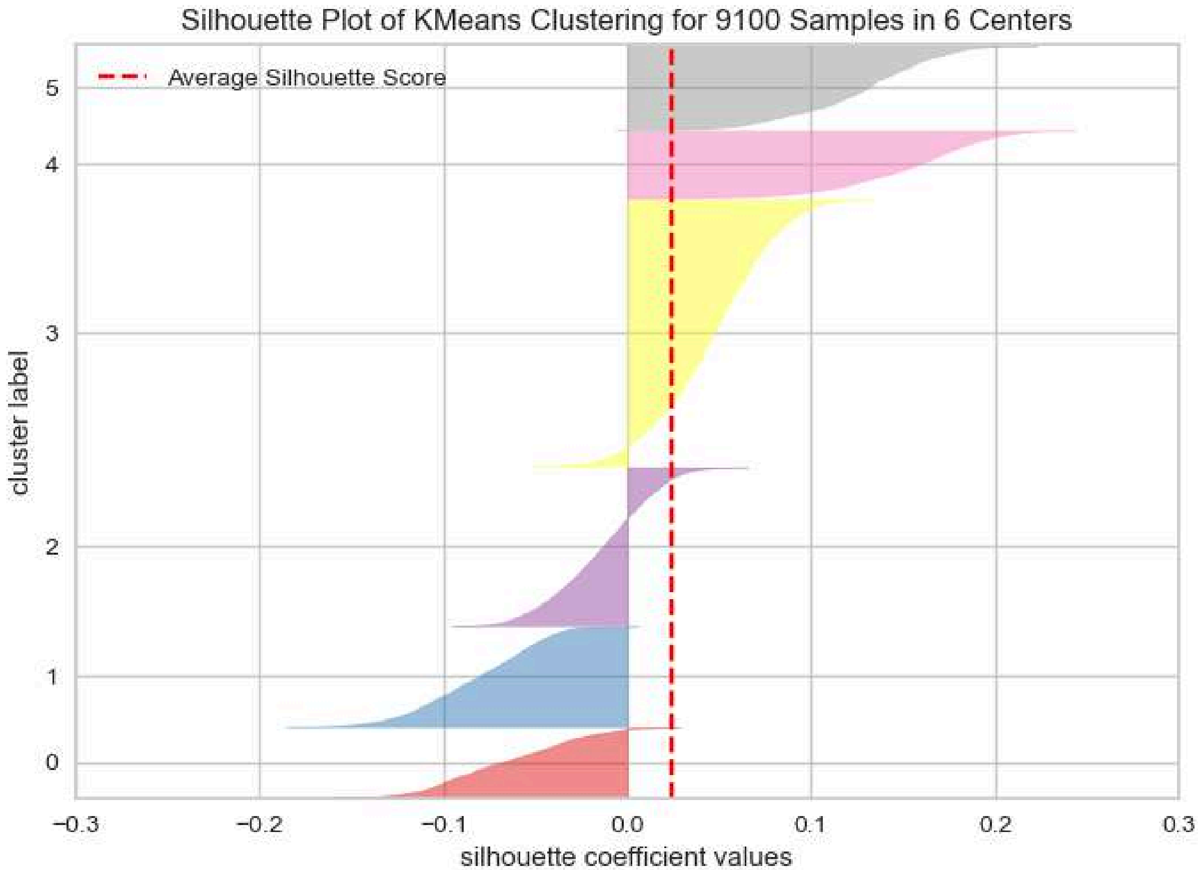
Out[34]:
```
<Axes: title={'center': 'Distortion Score Elbow for KMeans Clustering'}, xl
abel='k', ylabel='distortion score'>
```

In [35]:
```
visualizer = SilhouetteVisualizer(
    KMeans(5, random_state=42)
)
visualizer.fit(recipe_embeddings)
visualizer.show()
```



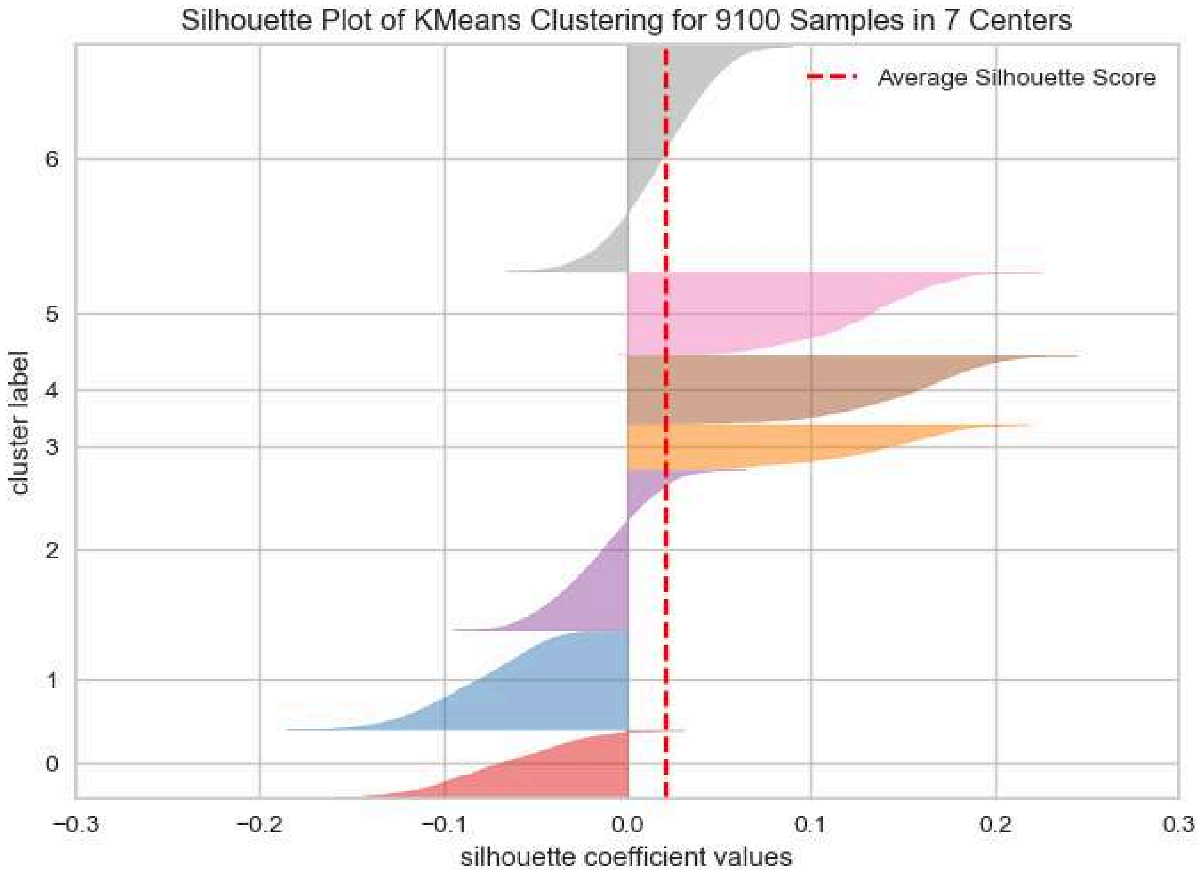
Out[35]: <Axes: title={'center': 'Silhouette Plot of KMeans Clustering for 9100 Samples in 5 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [36]:
```
visualizer = SilhouetteVisualizer(
    KMeans(6, random_state=42)
)
visualizer.fit(recipe_embeddings)
visualizer.show()
```



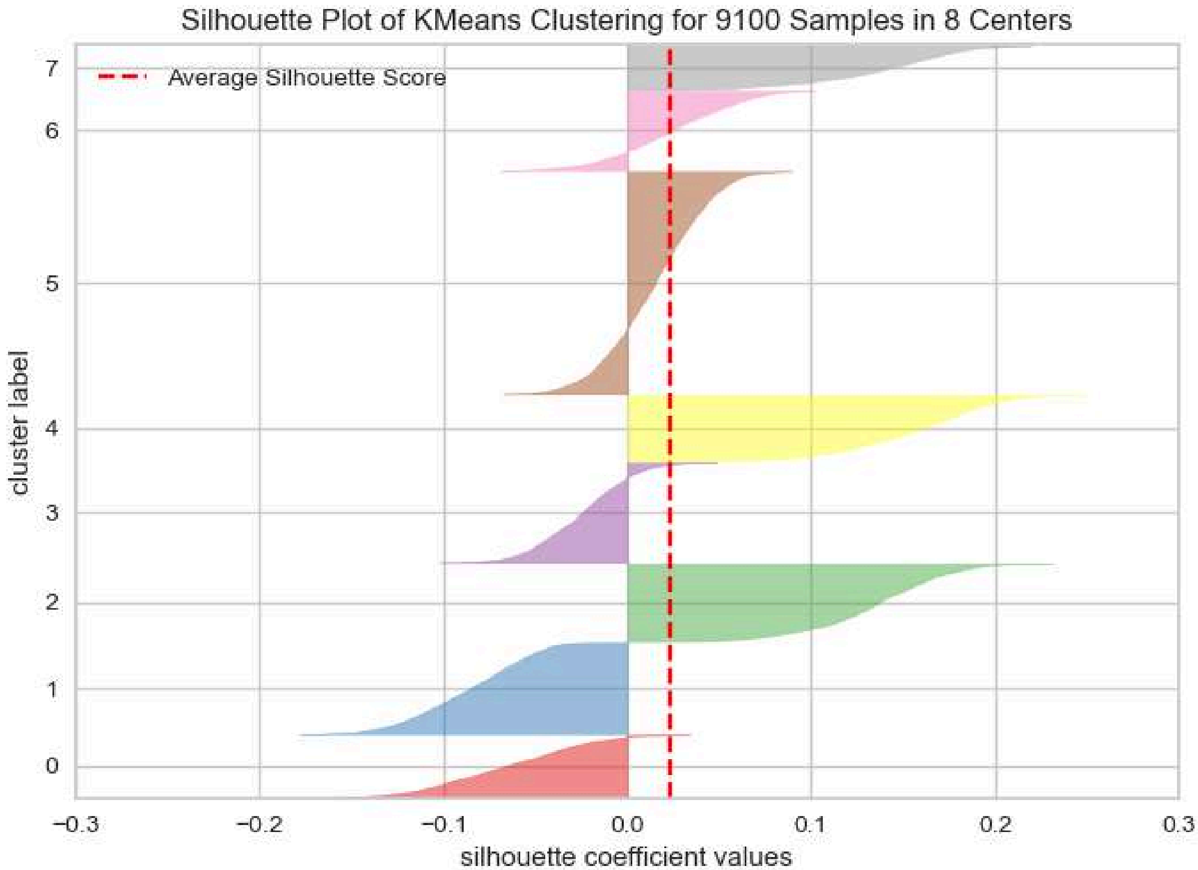
Out[36]: <Axes: title={'center': 'Silhouette Plot of KMeans Clustering for 9100 Samples in 6 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [37]:
```
visualizer = SilhouetteVisualizer(
    KMeans(7, random_state=42)
)
visualizer.fit(recipe_embeddings)
visualizer.show()
```



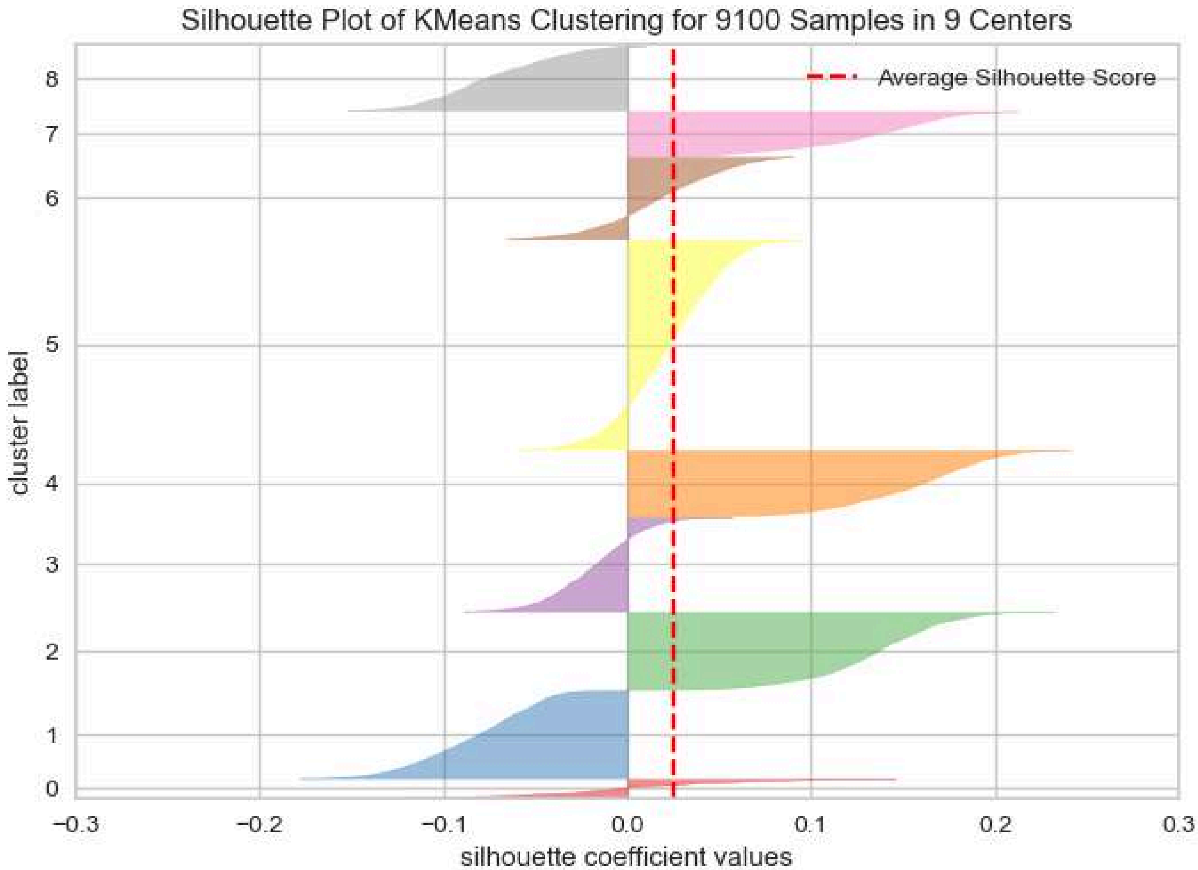
Out[37]: <Axes: title={'center': 'Silhouette Plot of KMeans Clustering for 9100 Samples in 7 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [38]:
```
visualizer = SilhouetteVisualizer(
    KMeans(8, random_state=42)
)
visualizer.fit(recipe_embeddings)
visualizer.show()
```



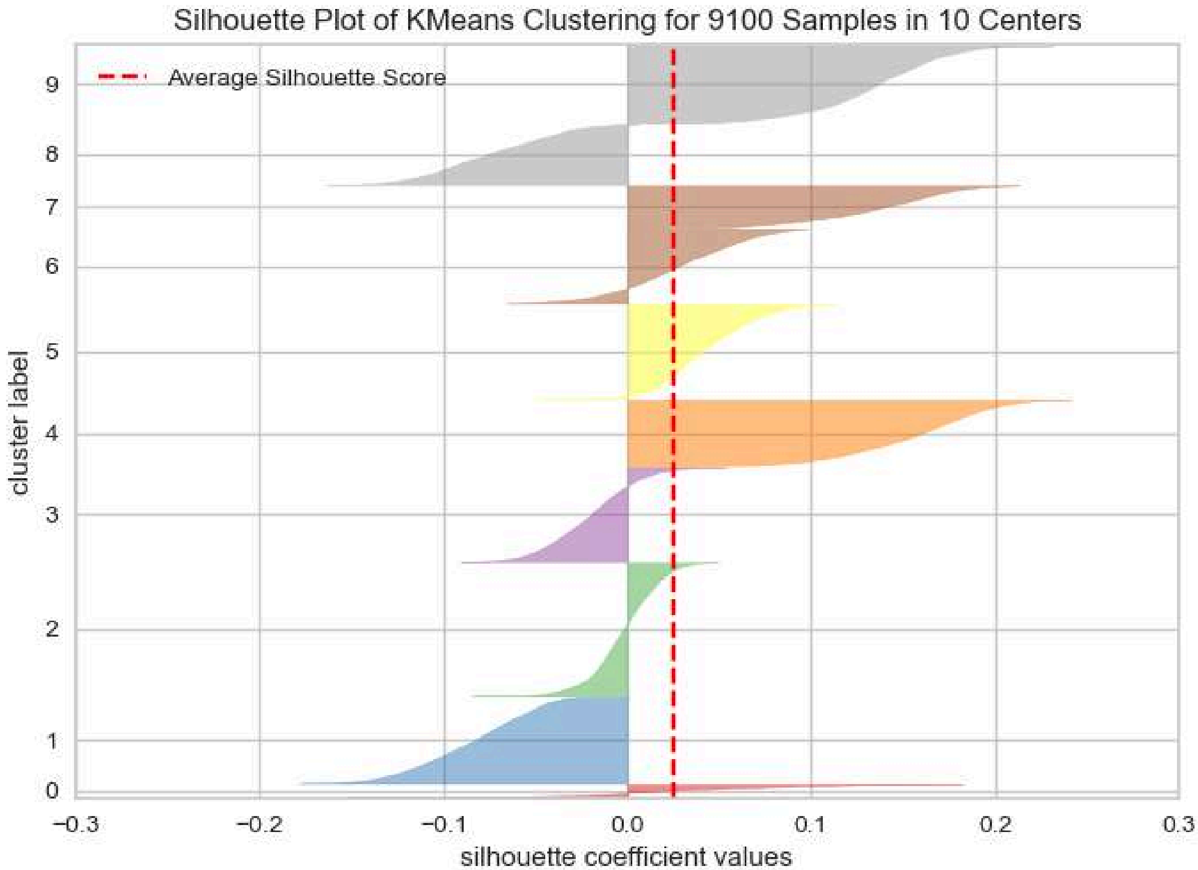
Out[38]: <Axes: title={'center': 'Silhouette Plot of KMeans Clustering for 9100 Samples in 8 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [39]:
```
visualizer = SilhouetteVisualizer(
    KMeans(9, random_state=42)
)
visualizer.fit(recipe_embeddings)
visualizer.show()
```



Out[39]: <Axes: title={'center': 'Silhouette Plot of KMeans Clustering for 9100 Samples in 9 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [40]:
```
visualizer = SilhouetteVisualizer(
    KMeans(10, random_state=42)
)
visualizer.fit(recipe_embeddings)
visualizer.show()
```

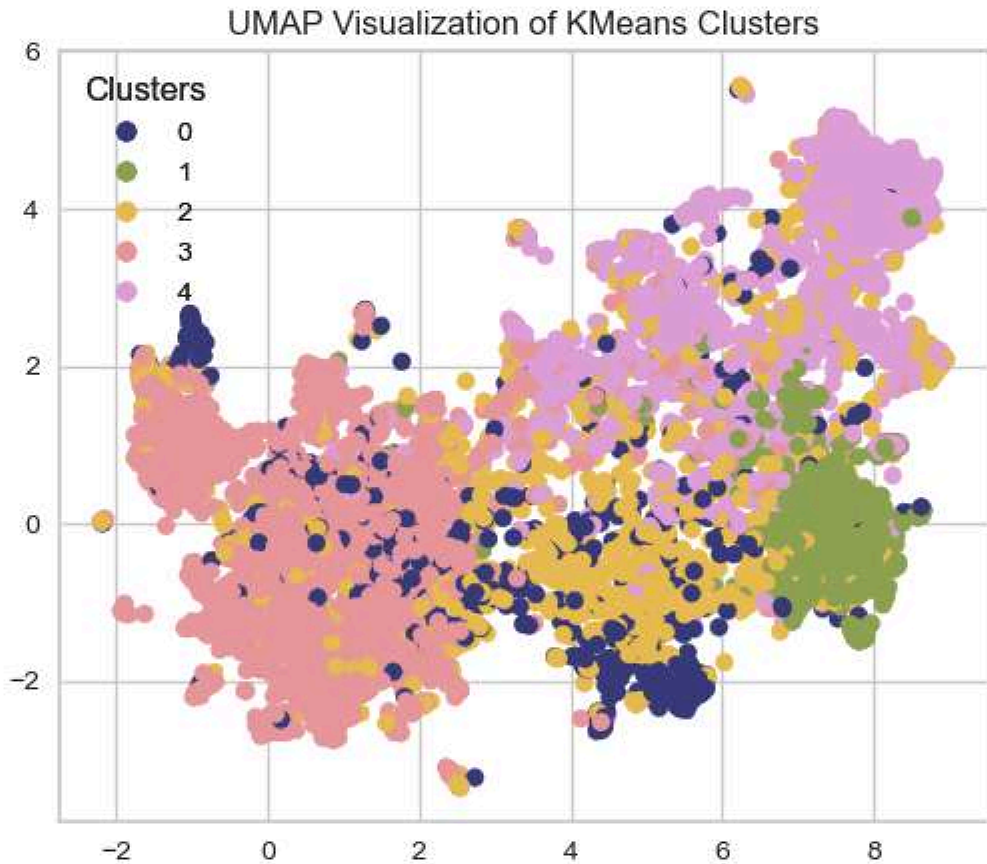


Out[40]: <Axes: title={'center': 'Silhouette Plot of KMeans Clustering for 9100 Samples in 10 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

In [41]:
```
optimal_k = 5
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_labels = kmeans.fit_predict(recipe_emb_df)

plot_umap_clusters(
    data=recipe_emb_df,
    cluster_labels=kmeans_labels,
    raw_sents=recipes_df["name"].tolist(),  # Original recipe names
    show_labels=False,
    size=40,
    n_neighbors=15,
    title="UMAP Visualization of KMeans Clusters"
)
```

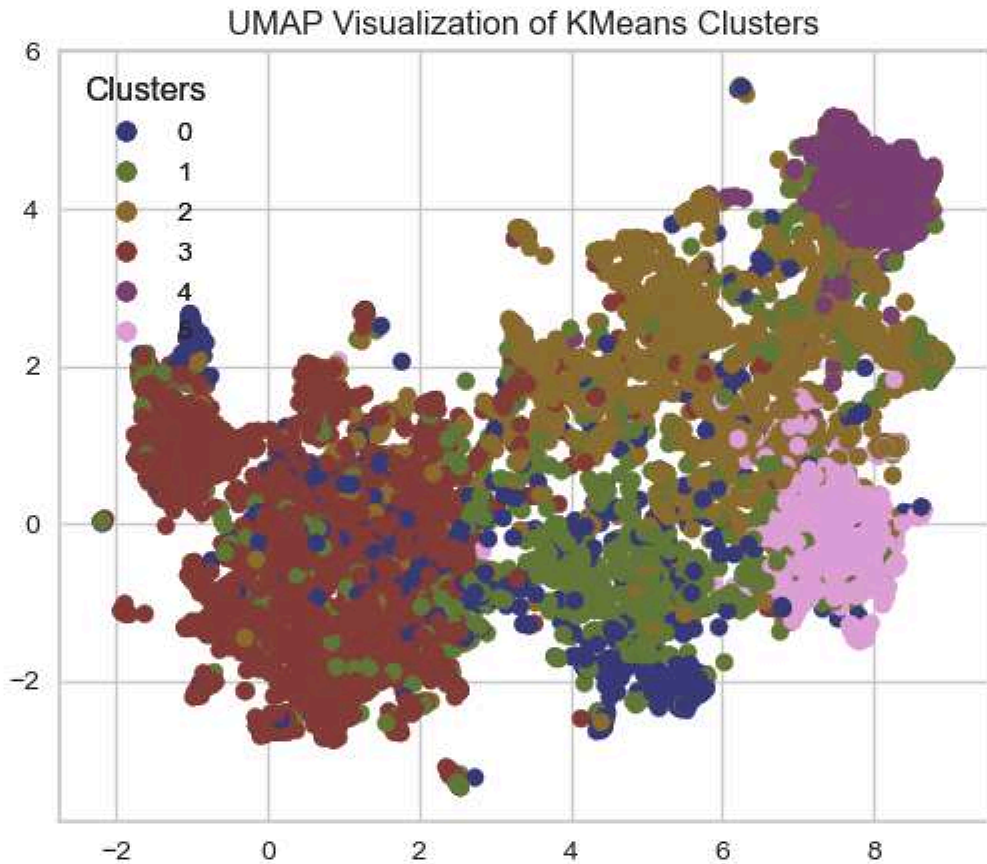
```
/opt/anaconda3/envs/cpsc330/lib/python3.12/site-packages/umap/umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no
seed for parallelism.
  warn(
```



In [42]:
```
optimal_k = 6
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_labels = kmeans.fit_predict(recipe_emb_df)

plot_umap_clusters(
    data=recipe_emb_df,
    cluster_labels=kmeans_labels,
    raw_sents=recipes_df["name"].tolist(),  # Original recipe names
    show_labels=False,
    size=40,
    n_neighbors=15,
    title="UMAP Visualization of KMeans Clusters"
)
```

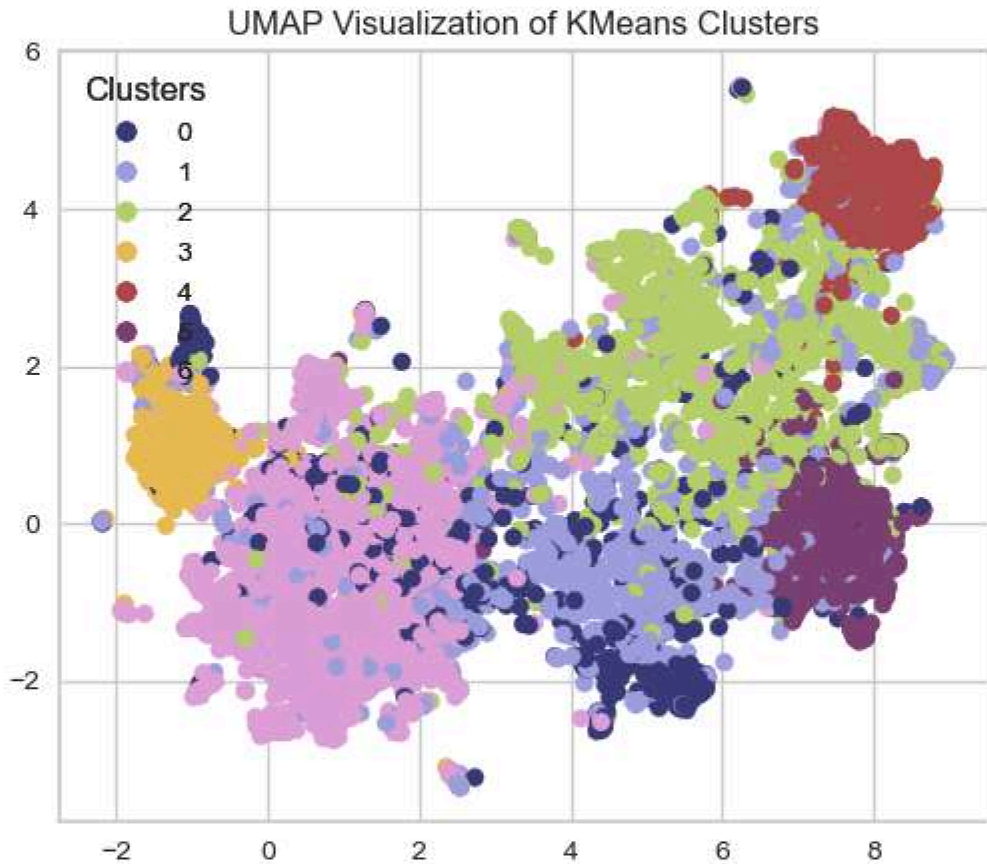
```
/opt/anaconda3/envs/cpsc330/lib/python3.12/site-packages/umap/umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no
seed for parallelism.
  warn(
```



In [43]:
```
optimal_k = 7
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_labels = kmeans.fit_predict(recipe_emb_df)

plot_umap_clusters(
    data=recipe_emb_df,
    cluster_labels=kmeans_labels,
    raw_sents=recipes_df["name"].tolist(),  # Original recipe names
    show_labels=False,
    size=40,
    n_neighbors=15,
    title="UMAP Visualization of KMeans Clusters"
)
```

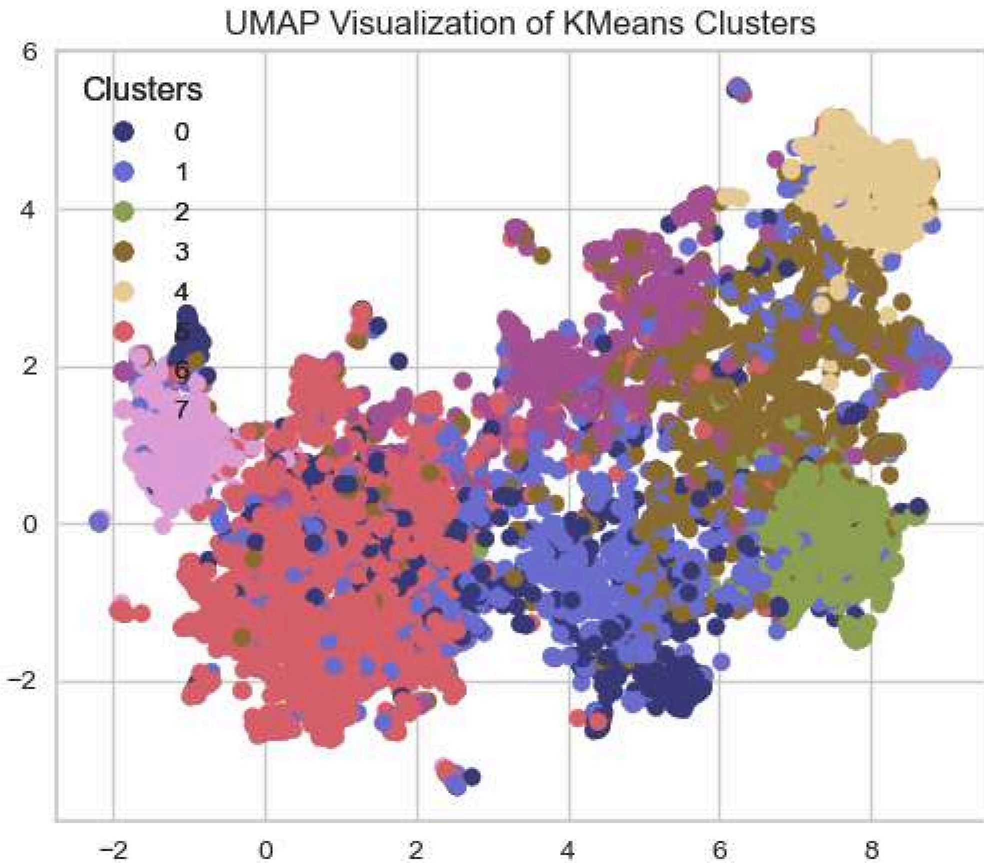
```
/opt/anaconda3/envs/cpsc330/lib/python3.12/site-packages/umap/umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no
seed for parallelism.
  warn(
```



In [44]:
```
optimal_k = 8
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_labels = kmeans.fit_predict(recipe_emb_df)

plot_umap_clusters(
    data=recipe_emb_df,
    cluster_labels=kmeans_labels,
    raw_sents=recipes_df["name"].tolist(),  # Original recipe names
    show_labels=False,
    size=40,
    n_neighbors=15,
    title="UMAP Visualization of KMeans Clusters"
)
```

```
/opt/anaconda3/envs/cpsc330/lib/python3.12/site-packages/umap/umap_.py:1952:
UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no
seed for parallelism.
  warn(
```



3.2 DBSCAN

rubric={points:6}

Your tasks:

1. Cluster recipe names using `DBSCAN` with `metric="cosine"`. Make some attempts to tune the hyperparameters `eps` and `min_samples`.

Solution_3.2

Points: 6

The DBSCAN results show that eps=0.1 and min_samples=7 gives us the best clustering quality, with a silhouette score of 0.3951 and two well-separated clusters. This is a good balance between cohesion and separation.

Lower eps values (e.g., 0.1) seem to result in better clustering results, while higher eps values (e.g., 0.2 or above) produced more clusters but resulted in lower or negative silhouette scores, which shows poor separation.

The combination of eps=0.5 and min_samples=7 resulted in only one cluster with most points marked as noise, suggesting overgeneralization. Therefore, eps=0.1 and min_samples=7 are best recommended for better clustering.

In [45]:

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score

# Define hyperparameter ranges to explore
eps_values = [0.1, 0.2, 0.3, 0.4, 0.5]
min_samples_values = [3, 5, 7]

# Function to perform DBSCAN with different hyperparameters
def perform_dbscan(data, eps_values, min_samples_values):
    for eps in eps_values:
        for min_samples in min_samples_values:
            # Perform DBSCAN clustering with cosine metric
            dbscan = DBSCAN(eps=eps, min_samples=min_samples, metric='cosine
            labels = dbscan.fit_predict(data)

            # Count unique labels and calculate the number of clusters (excl
            unique_labels = set(labels)
            num_clusters = len(unique_labels) - (-1 in unique_labels)  # Exc
            non_noise_mask = labels != -1

            # Ensure there are at least 2 clusters (excluding noise)
            if num_clusters > 1 and non_noise_mask.sum() > 1:
                # Calculate silhouette score for DBSCAN (ignoring noise poir
                score = silhouette_score(data[non_noise_mask], labels[non_nc
                print(f"DBSCAN with eps={eps}, min_samples={min_samples} -> 
            else:
                print(f"DBSCAN with eps={eps}, min_samples={min_samples} res

# Perform DBSCAN clustering and evaluate with enhanced handling
perform_dbscan(recipe_emb_df, eps_values, min_samples_values)
```

```
DBSCAN with eps=0.1, min_samples=3 -> Silhouette Score: 0.2818, Clusters: 20
DBSCAN with eps=0.1, min_samples=5 -> Silhouette Score: 0.3886, Clusters: 4
DBSCAN with eps=0.1, min_samples=7 -> Silhouette Score: 0.3951, Clusters: 2
DBSCAN with eps=0.2, min_samples=3 -> Silhouette Score: -0.0551, Clusters: 1
26
DBSCAN with eps=0.2, min_samples=5 -> Silhouette Score: -0.0127, Clusters: 3
5
DBSCAN with eps=0.2, min_samples=7 -> Silhouette Score: 0.0797, Clusters: 14
DBSCAN with eps=0.3, min_samples=3 -> Silhouette Score: -0.1709, Clusters: 7
3
DBSCAN with eps=0.3, min_samples=5 -> Silhouette Score: -0.0603, Clusters: 1
8
DBSCAN with eps=0.3, min_samples=7 -> Silhouette Score: -0.0572, Clusters: 1
2
DBSCAN with eps=0.4, min_samples=3 -> Silhouette Score: 0.0192, Clusters: 26
DBSCAN with eps=0.4, min_samples=5 -> Silhouette Score: 0.0592, Clusters: 8
DBSCAN with eps=0.4, min_samples=7 -> Silhouette Score: 0.1223, Clusters: 3
DBSCAN with eps=0.5, min_samples=3 -> Silhouette Score: 0.1093, Clusters: 4
DBSCAN with eps=0.5, min_samples=5 -> Silhouette Score: 0.1582, Clusters: 2
DBSCAN with eps=0.5, min_samples=7 resulted in too few clusters or all nois
e. Clusters: 1, Noise points: 458
```

3.3 Hierarchical clustering

rubric={points:6}

Your tasks:

1. Try hierarchical clustering with `metric="cosine"` on this problem. Show a dendrogram by using a suitable truncation method.
2. Create flat clusters by cutting the tree at the appropriate level.

> *Note: Try orientation="left" of `dendrogram` for better readability of the dendrogram.*

Solution_3.3

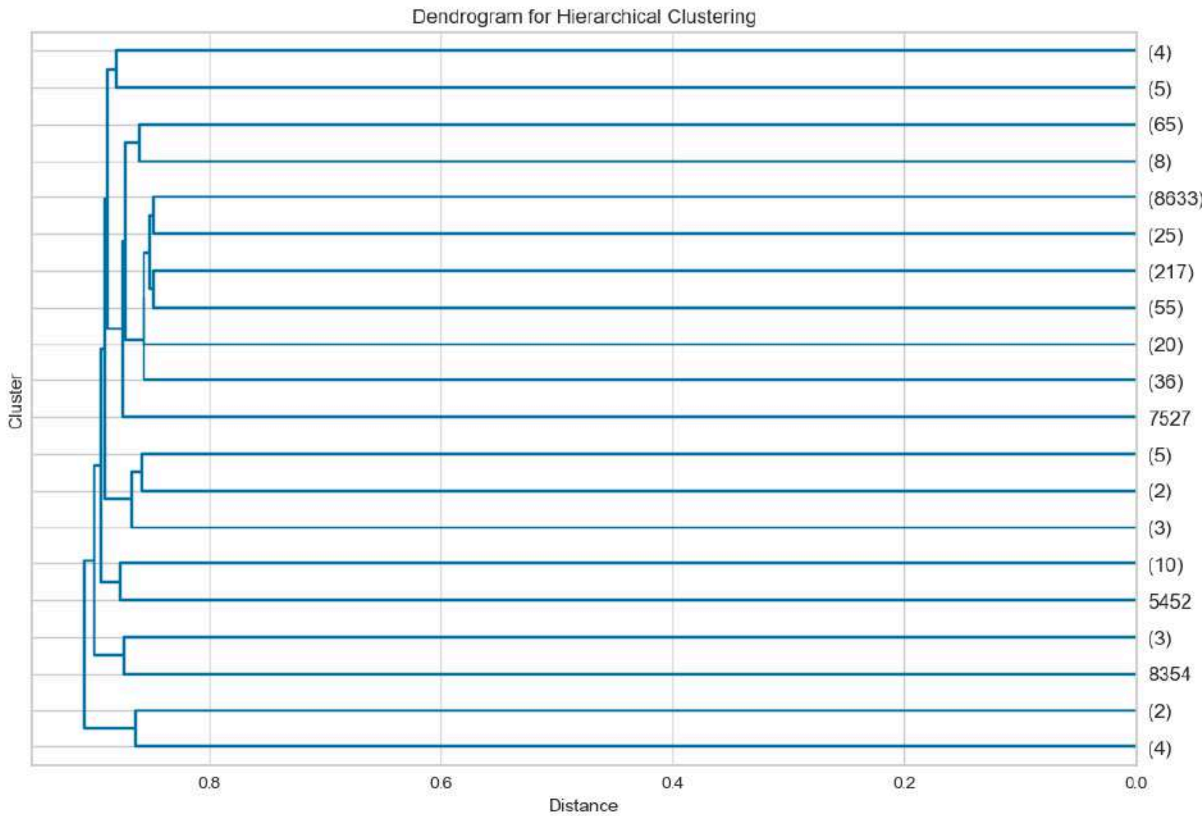
Points: 6

Type your answer here, replacing this text.

In [46]:
```
linkage_matrix = linkage(recipe_emb_df, method='average', metric='cosine')

plt.figure(figsize=(12, 8))
dendrogram(linkage_matrix, truncate_mode='lastp', orientation='left', p=20)
plt.title("Dendrogram for Hierarchical Clustering")
plt.xlabel("Distance")
```

```
plt.ylabel("Cluster")
plt.show()
```



Looking at the dendrogram and considering a distance threshold around 0.8-0.9, we should choose 4-5 clusters for optimal recipe categorization. The dendrogram shows 4-5 major branches at this threshold level, which is supported by the cluster sizes shown in the parentheses on the right (including one large cluster of 8633 recipes and several smaller but substantial clusters). This clustering provides a balanced distribution of recipes while maintaining meaningful distinctions between different types of recipes. 5 clusters is also practical from a user perspective, as it's enough to capture major recipe categories (like main dishes, desserts, sides, beverages, and snacks) without becoming too granular or too broad in its categorization.

In [47]:
```
from sklearn.cluster import AgglomerativeClustering
n_clusters = 5
hierarchical = AgglomerativeClustering(n_clusters=n_clusters, metric='cosine
hierarchical_labels = hierarchical.fit_predict(recipe_emb_df)

print(f"Number of clusters formed: {n_clusters}")
```

```
Number of clusters formed: 5
```

In [48]:
```
labels_series = pd.Series(hierarchical_labels)
counts = labels_series.value_counts()
print(counts)
```

```
0    9069
1    11
4    10
3    6
2    4
Name: count, dtype: int64
```

3.4 Manual interpretation of clusters

rubric={points:6}

Your tasks:

1. Label the topics/themes you see in the clusters created by different clustering methods.
2. Do you see a common theme across clusters created by different clustering methods? Do you see any differences between the clusters created by different clustering methods?

Solution_3.4

Points: 6

1.1 When we create 5 clusters, we have 5 categories in the recipes:

- Main dishes
- Desserts
- Sides
- Beverages
- Snacks

1.2 When we create 7 clusters (using Kmeans), we have 7 categories in the recipes:

- Main dishes
- Appetizer
- Cake
- Bakery (muffins, pies, puffs)
- Beverages
- Soup
- Dip and Sauce

2. Common themes across clustering methods: All clustering approaches capture the fundamental recipe categories, particularly main dishes and beverages, which are distinct categories regardless of the number of clusters. The key food preparation

types (cooking vs. baking) are also preserved across both methods. This consistency suggests these are natural, well-separated categories in the recipe dataset.

Differences between clustering methods: The 7-cluster approach provides more granular categorization, particularly in breaking down the broader categories from the 5-cluster method. For example, the general "Desserts" category in the 5-cluster method is split into more specific "Cake" and "Bakery" categories in the 7-cluster method. This shows how increasing the number of clusters allows for more specific culinary categories while the 5-cluster approach is simpler, and has more general categorization.

Before submitting your assignment, please make sure you have followed all the instructions in the Submission instructions section at the top.