Clustering

Problem 1: Segment Customers

The problem that we are going to solve in this assignment is to segment customers into different groups based on their shopping trends.

In [27]:
```
# import packages
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import AgglomerativeClustering
```

Load data

Our dataset has five columns: CustomerID, Genre, Age, Annual Income, and Spending Score. To view the results in two-dimensional feature space, we will retain only two of these five columns. We can remove CustomerID column, Genre, and Age column. We will retain the Annual Income (in thousands of dollars) and Spending Score (1-100) columns. The Spending Score column signifies how often a person spends money in a mall on a scale of 1 to 100 with 100 being the highest spender.

In [28]:
```python
# Load the data
shopping_data = pd.read_csv('https://raw.githubusercontent.com/zariable/data/master/shopping_data.csv')
shopping_data.rename(
    columns={
        'CustomerID': 'customer_id',
        'Genre': 'genre',
        'Age': 'age',
        'Annual Income (k$)': 'annual_income',
        'Spending Score (1-100)': 'spending_score'
    },
    inplace=True
)
display(shopping_data.head())

# TODO: retain only anual_income and spending_score for clustering

subset = shopping_data.loc[:,['annual_income','spending_score']]
print(subset.head())
```

	customer_id	genre	age	annual_income	spending_score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
   annual_income  spending_score
0             15              39
1             15              81
2             16               6
3             16              77
4             17              40
```

Hierarchical Clustering

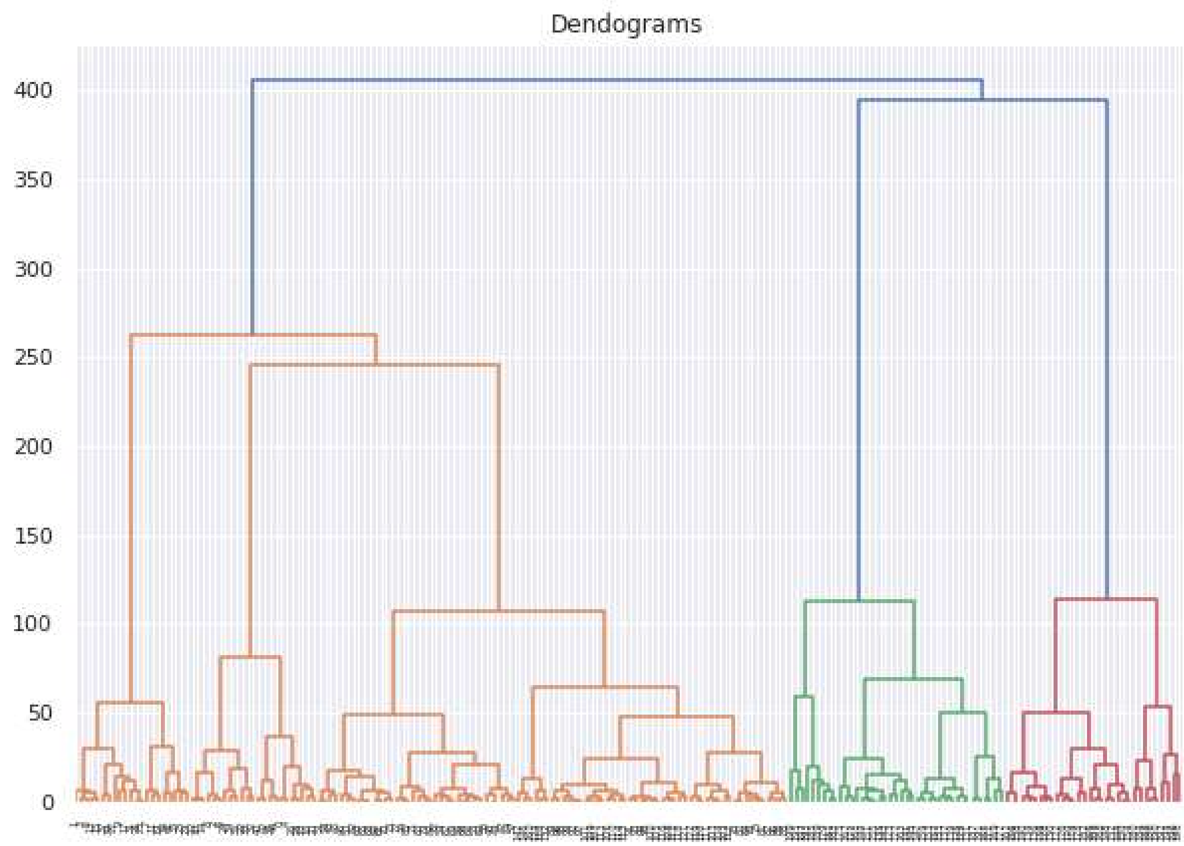
First, we will apply hierarchical clustering and use dendrogram to help find the number of clusters within the data.

TODO: Use dendrogram to plot hierarchical clustering and find the number of clusters that makes sense.

In [29]:
```
from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n_clusters=4, affinity='euclidean', linkage='war
d')
y_ward = hc.fit_predict(subset)

y_ward
```

Out[29]:
```
array([0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3,
       0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 1, 2, 1, 2,
       0, 2, 1, 2, 1, 2, 1, 2, 1, 2, 0, 2, 1, 2, 0, 2, 1, 2, 1, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 0, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2,
       1, 2])
```

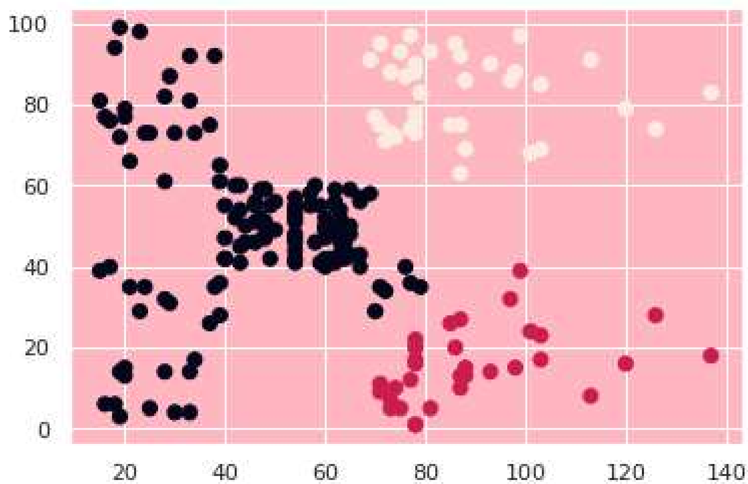
In [30]:
```
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))
plt.title("Dendograms")
dend = shc.dendrogram(shc.linkage(subset, method='ward'))
```



Number of optimal clusters is 3

TODO: Apply hierarchical clustering based on the number of clusters you pick from the dendrogram, and visualize the results using scatterplot.

In [31]:
```
# TODO
hc_2 = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='wa
rd')
z_ward = hc_2.fit_predict(subset)
z_ward
fig, ax = plt.subplots()
ax.set_facecolor('lightpink')
plt.scatter(subset["annual_income"],subset["spending_score"],s=50,c=z_ward)
plt.show()
```

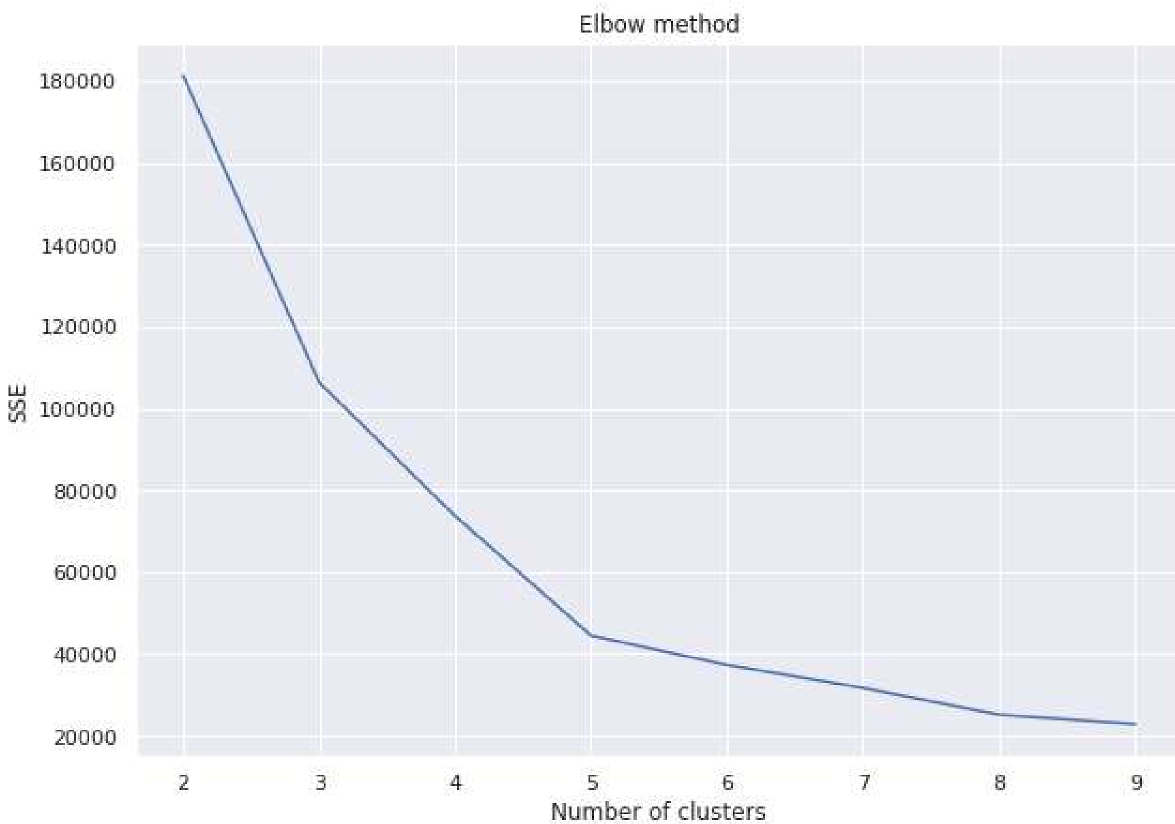


K-means Clustering

Then, we will apply k-means to the same data and visualize the results.

TODO: Vary the number of K from 2 to 10 and plot the Sum of Squared Error (SSE) as K increases and pick up the value of K that makes sense.

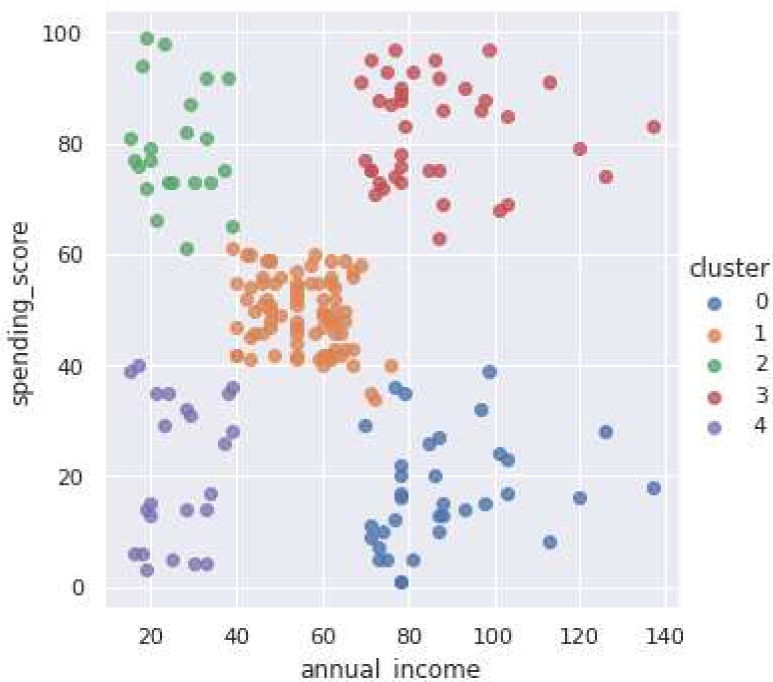
In [32]:
```
from sklearn.cluster import KMeans
# TODO
SSE = []
for i in range(2,10):
  kmeans = KMeans(n_clusters = i,init='random',random_state = 50)
  kmeans.fit(subset)
  SSE.append(kmeans.inertia_)

plt.figure(figsize=(10,7))
plt.plot(range(2,10),SSE)
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```



From the elbow curve, we can see that the number of optimal clusters is 5

TODO: Cluster the data using K-means based on the pre-defined value of K from the previous step and and visualize the results using scatterplot.

In [33]:
```
# TODO
kmeans = KMeans(n_clusters = 5,random_state = 50)
kmeans.fit(subset)
pred=kmeans.predict(subset)
subset['cluster']=pd.DataFrame(pred,columns=['Cluster'])
subset.head()
sns.lmplot(x='annual_income',y='spending_score',data=subset,fit_reg=False,
           hue='cluster',legend=True)
plt.show()
```



Problem 2: Clustering (Manually)

For the following dataset, perform the clustering “by hand”:

17 28 50 60 80 89 150 167 171 189

1. Use the K-means algorithm with K= 3 to cluster the data
2. Use hierarchical agglomerative clustering with single linkage to cluster the data
3. Use hierarchical agglomerative clustering with complete linkage to cluster the data
4. For K-means What will the final clusters be after 3 iterations if k=3 and the initial centers are 150, 171 and 189

In [34]:
```
#  1) Use the K-means algorithm with K= 3 to cluster the data

data = [17, 28, 50, 60, 80, 89, 150, 167, 171, 189]
k = 3
#Calculating distance
def dist(data_point, centroid):
    return np.sqrt(np.sum(data_point-centroid)**2)

#Creating initial clusters and assigning data points
def create_clusters(data, centroids):
    clusters = {i: [] for i in range(k)}
    # for i in range(k):
    #  clusters[i] = []
    for data_point in data:
      x=[]
      for centroid in centroids:
        x.append(dist(data_point, centroid))
        min_value_index = np.argmin(x)
      clusters[min_value_index].append(data_point)
    return clusters

#Calculating new centroids
def update_centroids(clusters):
    centroids = []
    for i in range(k):
        if clusters[i]:
            centroids.append(np.mean(clusters[i]).round())
        else:
            centroids.append(data[np.random.randint(len(data))])
    return centroids

# Initialize initial centroids randomly
centroids = []
random_index = [np.random.randint(len(data)) for i in range(k)]
for j in random_index:
    centroids.append(data[j])
print("initial centroids",centroids,type(centroids),"\n")

# Check if old and new centroids are same/different
while True:
    # Assign data points to clusters
    clusters = create_clusters(data, centroids)
    print("clusters",clusters)
    # Update centroids
    new_centroids = update_centroids(clusters)
    print("new_centroids",new_centroids,"\n")
    # compare centroids
    if np.array_equal(centroids, new_centroids):
        break

    centroids = new_centroids
# Print final clusters
for i, cluster in clusters.items():
    print(f"Cluster {i}: {cluster}")
```

```
initial centroids [171, 80, 50] <class 'list'>

clusters {0: [150, 167, 171, 189], 1: [80, 89], 2: [17, 28, 50, 60]}
new_centroids [169.0, 84.0, 39.0]

clusters {0: [150, 167, 171, 189], 1: [80, 89], 2: [17, 28, 50, 60]}
new_centroids [169.0, 84.0, 39.0]

Cluster 0: [150, 167, 171, 189]
Cluster 1: [80, 89]
Cluster 2: [17, 28, 50, 60]
```

In [35]:
```
# 2) Use hierarchical agglomerative clustering with single linkage to cluster
the data

data = [17, 28, 50, 60, 80, 89, 150, 167, 171, 189]
clusters=[]

#Calculating distance
def dist(x1,x2):
     return np.sqrt(np.sum(x1-x2)**2)
     #return abs(x1- x2)

#Finding minimum distance between members of two clusters
def min_linkage(cluster_1,cluster_2):
  if(len(cluster_1)==1 and len(cluster_2)== 1):
     return(dist(cluster_1[0],cluster_2[0]))

  else:
    min_distance = 1000
    for x1 in cluster_1:
      for x2 in cluster_2:
        distance = dist(x1,x2)
        if distance < min_distance:
            min_distance = distance
    return min_distance

#Assigning clusters
def create_clusters(data):
    for datapoint in data:                  #Creating clusters for each point i
n data
        clusters.append([datapoint])
    print("initial clusters",clusters,type(clusters),range(len(clusters)))
    while len(clusters)>1:
        min_distance = 1000               # Assigning a maximum threshold depe
nding on the values in data(if the values in the dataset are unknown/large, we
could set the min_distance to infinity)
        min_j = None
        for i in range(len(clusters)):
           # print('i',clusters[i])
            for j in range(i+1,len(clusters)):
              #print('j',clusters[j])
              distance = min_linkage(clusters[i],clusters[j])
              if distance < min_distance:
                    min_distance = distance
                    min_j = j
            print("distance between",clusters[i],"and",clusters[min_j],"is",mi
n_distance, "\n")
            new_cluster = clusters[i]+clusters[min_j]
            clusters.remove(clusters[i])     #removing old clusters and append
ing new clusters
            clusters.remove(clusters[min_j-1])
            clusters.append(new_cluster)
            print("new_cluster",clusters)
            break

create_clusters(data)
```

```
initial clusters [[17], [28], [50], [60], [80], [89], [150], [167], [171], [1
89]] <class 'list'> range(0, 10)
distance between [17] and [28] is 11.0

new_cluster [[50], [60], [80], [89], [150], [167], [171], [189], [17, 28]]
distance between [50] and [60] is 10.0

new_cluster [[80], [89], [150], [167], [171], [189], [17, 28], [50, 60]]
distance between [80] and [89] is 9.0

new_cluster [[150], [167], [171], [189], [17, 28], [50, 60], [80, 89]]
distance between [150] and [167] is 17.0

new_cluster [[171], [189], [17, 28], [50, 60], [80, 89], [150, 167]]
distance between [171] and [150, 167] is 4.0

new_cluster [[189], [17, 28], [50, 60], [80, 89], [171, 150, 167]]
distance between [189] and [171, 150, 167] is 18.0

new_cluster [[17, 28], [50, 60], [80, 89], [189, 171, 150, 167]]
distance between [17, 28] and [50, 60] is 22.0

new_cluster [[80, 89], [189, 171, 150, 167], [17, 28, 50, 60]]
distance between [80, 89] and [17, 28, 50, 60] is 20.0

new_cluster [[189, 171, 150, 167], [80, 89, 17, 28, 50, 60]]
distance between [189, 171, 150, 167] and [80, 89, 17, 28, 50, 60] is 61.0

new_cluster [[189, 171, 150, 167, 80, 89, 17, 28, 50, 60]]
```

In [36]:
```
        # 4) For K-means What will the final clusters be after 3 iterations if
k=3 and the initial centers are 150, 171 and 189

data = [17, 28, 50, 60, 80, 89, 150, 167, 171, 189]
k = 3
max_iter = 3

def dist(data_point, centroid):
    return np.sqrt(np.sum(data_point-centroid)**2)

def create_clusters(data, centroids):

    clusters = {i: [] for i in range(k)}
    for data_point in data:
      x=[]
      for centroid in centroids:
        x.append(dist(data_point, centroid))
        min_value_index = np.argmin(x)
      clusters[min_value_index].append(data_point)
    return clusters

def update_centroids(clusters):
    centroids = []
    for i in range(k):
        if clusters[i]:
            centroids.append(np.mean(clusters[i]).round())
        else:
            centroids.append(data[np.random.randint(len(data))])
    return centroids

# Initialize centroids [150,171,189]

centroids = [150,171,189]
print("initial centroids",centroids,type(centroids),"\n")

# Check if old and new centroids are same/different
for i in range(max_iter):
    # Assign data points to clusters
    clusters = create_clusters(data, centroids)
    print("new_cluster",clusters)
    # Update centroids
    new_centroids = update_centroids(clusters)
    print("new_centroids",new_centroids,"\n")
    # compare centroids
    if np.array_equal(centroids, new_centroids):
        break

    centroids = new_centroids

# Print final clusters
print("cluster after 3 iteration")
for i, cluster in clusters.items():
    print(f"Cluster {i}: {cluster}")
```

```
initial centroids [150, 171, 189] <class 'list'>

new_cluster {0: [17, 28, 50, 60, 80, 89, 150], 1: [167, 171], 2: [189]}
new_centroids [68.0, 169.0, 189.0]

new_cluster {0: [17, 28, 50, 60, 80, 89], 1: [150, 167, 171], 2: [189]}
new_centroids [54.0, 163.0, 189.0]

new_cluster {0: [17, 28, 50, 60, 80, 89], 1: [150, 167, 171], 2: [189]}
new_centroids [54.0, 163.0, 189.0]

cluster after 3 iteration
Cluster 0: [17, 28, 50, 60, 80, 89]
Cluster 1: [150, 167, 171]
Cluster 2: [189]
```

Final cluster after 3 iterations for centroids [150,171,189] :

Cluster 0: [17, 28, 50, 60, 80, 89] Cluster 1: [150, 167, 171] Cluster 2: [189]

Bonus points

Use the dataset of accepted papers at the AAAI 2014 conference to find clusters of papers using K-Means. You can use paper title or abstract to build your features using [Bag of Words (https://en.wikipedia.org/wiki/Bag-of-words_model)](https://en.wikipedia.org/wiki/Bag-of-words_model).

1. Vary the number of K from 2 to 6 and show if the results vary and assess the clusters obtained.
2. Make a case regarding which clusters 'make sense' e.g., is there a cluster were papers on reinforcement learning are together vs. another cluster which has papers on deep learning.

In [40]:
```python
nltk.download("stopwords")
nltk.download("punkt")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[40]:
```
True
```

In [41]:
```
# Load the Relevant Libraries
import sklearn as sk
import nltk
import re
import string
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.feature_extraction.text import CountVectorizer

# URL for the AAAI (UW Repository)
aaai_data = pd.read_csv("https://raw.githubusercontent.com/zariable/data/maste
r/AAAI2014AcceptedPapers.csv")
#aaai_data.head()
title = aaai_data['title']


def clean_text(text):

    text = text.lower()  # Lowercase words
    text = re.sub(r"\[(.*?)\]", "", text)  # Remove [+XYZ chars] in content
    text = re.sub(r"\s+", " ", text)  # Remove multiple spaces in content
    text = re.sub(r"(?<=\w)-(?=\w)", " ", text)  # Replace dash between words
    text = text.translate(str.maketrans('', '', string.punctuation))  # Remove
punctuation

    stop_words = set(stopwords.words('english'))
    tokens = word_tokenize(text)  # Get tokens from text
    tokens = [t for t in tokens if not t in stop_words]  # Remove stopwords
    text = ' '.join(tokens)
    return text

title = title.apply(clean_text)

# Create a Bag of Words representation
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(title)

# Apply K-Means clustering
for k in range(2,7):
  kmeans = KMeans(n_clusters=k)
  y_kmeans = kmeans.fit_predict(X)

  print(f'Number of clusters: {k}')
  for i, centroid in enumerate(kmeans.cluster_centers_):
        top_terms = [vectorizer.get_feature_names_out()[index] for index in ce
ntroid.argsort()[:-10:-1]] #getting the top 10 terms for each cluster
        print(f'Cluster {i}: {" / ".join(top_terms)}')
  print()
```

```
Number of clusters: 2
Cluster 0: based / model / data / domain / learning / planning / information
/ modeling / optimization
Cluster 1: learning / using / multi / social / via / search / planning / mode
l / online

Number of clusters: 3
Cluster 0: rcc / calculi / partitioning / constants / consistency / extended
/ checking / networks / fast
Cluster 1: based / using / planning / search / model / social / via / analysi
s / information
Cluster 2: learning / multi / transfer / instance / sparse / view / image / o
nline / based

Number of clusters: 4
Cluster 0: supervised / image / via / sparse / classification / low / rank /
analysis / view
Cluster 1: learning / multi / instance / transfer / sparse / models / online
/ based / robust
Cluster 2: data / modeling / based / mining / quality / occupant / risk / sur
veillance / planning
Cluster 3: based / using / search / planning / model / social / information /
games / linear

Number of clusters: 5
Cluster 0: monte / carlo / using / embedding / hamiltonian / view / neural /
pre / side
Cluster 1: social / networks / analysis / choice / using / learning / randomi
zed / temporal / recommender
Cluster 2: based / using / model / planning / search / via / games / informat
ion / linear
Cluster 3: learning / transfer / based / sparse / classification / online / b
ayesian / models / data
Cluster 4: multi / learning / view / task / via / instance / agent / robot /
dictionary

Number of clusters: 6
Cluster 0: group / sparsity / multiple / user / recommendation / online / ite
ratively / reweighted / dynamics
Cluster 1: based / model / domain / data / learning / planning / games / info
rmation / agent
Cluster 2: social / networks / choice / using / analysis / temporal / aware /
influence / neural
Cluster 3: using / search / planning / model / via / linear / multi / informa
tion / supervised
Cluster 4: learning / multi / instance / transfer / sparse / models / via / v
iew / online
Cluster 5: sentiment / analysis / commonsense / common / knowledge / acquirin
g / applications / compositionality / computation
```

For this particular iteration, I think k=3 makes more sense because cluster 0 has more terms related to game theory and cluster 3 in k=6 has more terms related to deep learning. But since the centroids keep changing, I think we have to average the results of multiple runs to get more stable set of clusters.

3. use hierarchical agglomerative clustering with Complete linkage to cluster the data.

Step 1 :- Assume each data point to be its own cluster

Initial cluster [17] [28] [50] [60] [80] [89] [150] [167] [171] [189]

Step 2 :- we calculate the distance b/w all the pairs of clusters and find the min distance between 2 data points.

-> In this case cluster 161 and 171 has a min distance of 4 so we merge this into one cluster

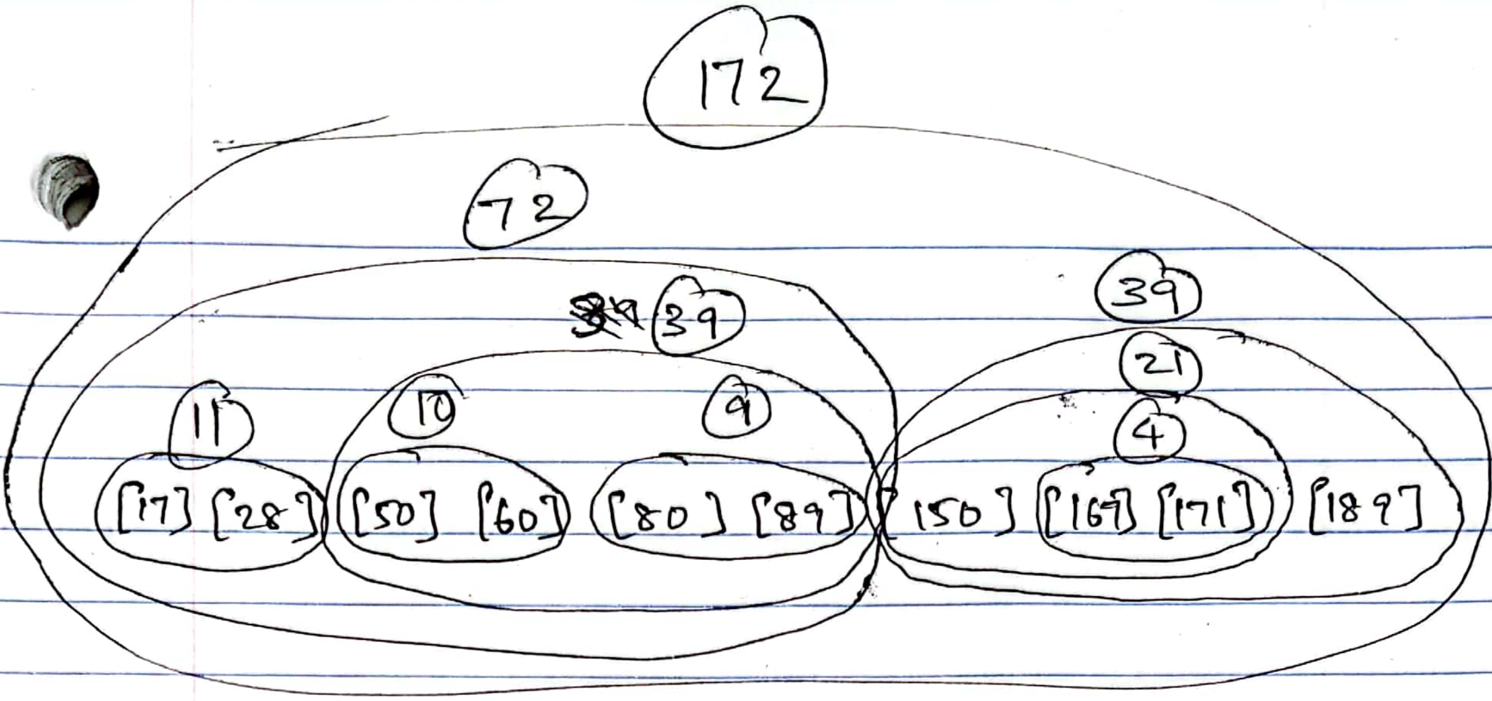
Iteration 1 [17] [28] [50] [60] [80] [89] [150] [167, 171] [189]

Step 3 :- we recalculate the distance matrix. Since this is complete-linkage we calculate the max distance between this cluster & other data points

In Iteration 2, we combine the cluster with next minimum distance 80 and 89.

Iteration 2 [17] [28] [50] [60] [80, 89], [150], [167, 171], [189]

Step 4 :- we repeat the iterations till there is convergence and one single cluster is obtained



Final cluster

[17, 28, 50, 60, 80, 89, 150, 167, 171, 189]

2.) use hierarchical agglomerative clustering with single linkage to cluster the data

Step 1: Assume each data point to be its own cluster

Initial cluster [17], [28], [50], [60], [80], [89], [150], [167], [171], [189]

Step 2: we calculate the distance between all pairs of clusters using the single linkage method which considers the minimum distance between any 2 points in this cluster

→ In this case cluster 167 and 171 have a min distance 4

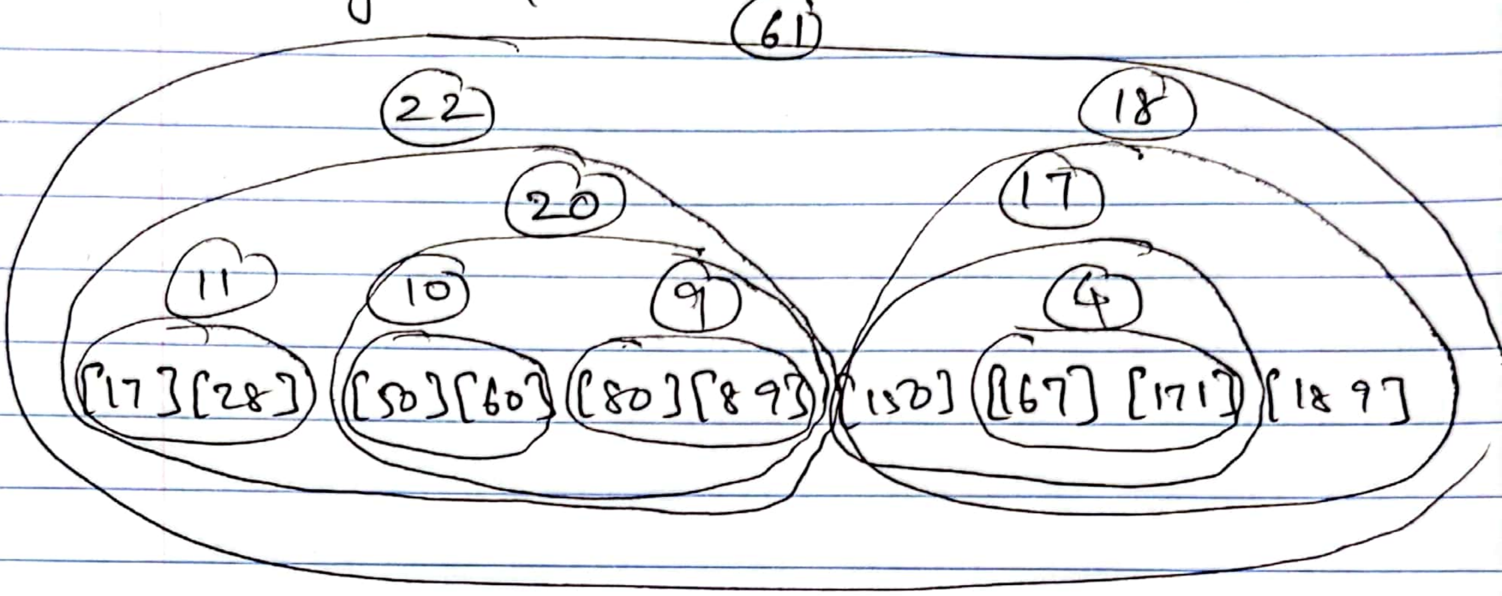
Iteration 1 [17] [28] [50] [60] [80] [89] [150] [167, 171] [189]

Step 3: Since this is single linkage we recalculate the ~~di~~ min distance between cluster [167, 171] and other data points. then merge the clusters that has the min distance

Iteration 2: [17] [28] [50] [60] [80, 89], [150] [167, 171] [189]

In this iteration we combined the clusters with next min distance 9.

Step 4: we repeat the steps till there is Convergence & one cluster is obtained



Final cluster

[17, 28, 50, 60, 80, 89, 150, 167, 171, 189]