Cluster Analysis, ANN, & Text Mining: Parts 1 - Clustering

This section will cover part 1 of the rubric, which includes K-means clustering and Hierarchical clustering.

K-Means Clustering - Rahul Gupta

We'll use K-Means clustering to group movies by similar IMDB rating, critics' score, and audience rating.

Reading the Dataset

First, we need to load the csv dataset into a python dataframe. We will also perform a bit of preprocessing by dropping any rows with missing information.

In [49]:
```
import pandas as pd

df = pd.read_csv('../CSC177/imdb_dataset.csv')

# Remove missing values
data = df.dropna()

data.head()
```

Out[49]:

	Unnamed: 0	title	title_type	genre	runtime	mpaa_rating	studio	thtr_rel_year	tht
0	1	Filly Brown	Feature Film	Drama	80.0	R	Indomina Media Inc.	2013	
1	2	The Dish	Feature Film	Drama	101.0	PG-13	Warner Bros. Pictures	2001	
2	3	Waiting for Guffman	Feature Film	Comedy	84.0	R	Sony Pictures Classics	1996	
3	4	The Age of Innocence	Feature Film	Drama	139.0	PG	Columbia Pictures	1993	
4	5	Malevolence	Feature Film	Horror	90.0	R	Anchor Bay Entertainment	2004	

5 rows × 33 columns

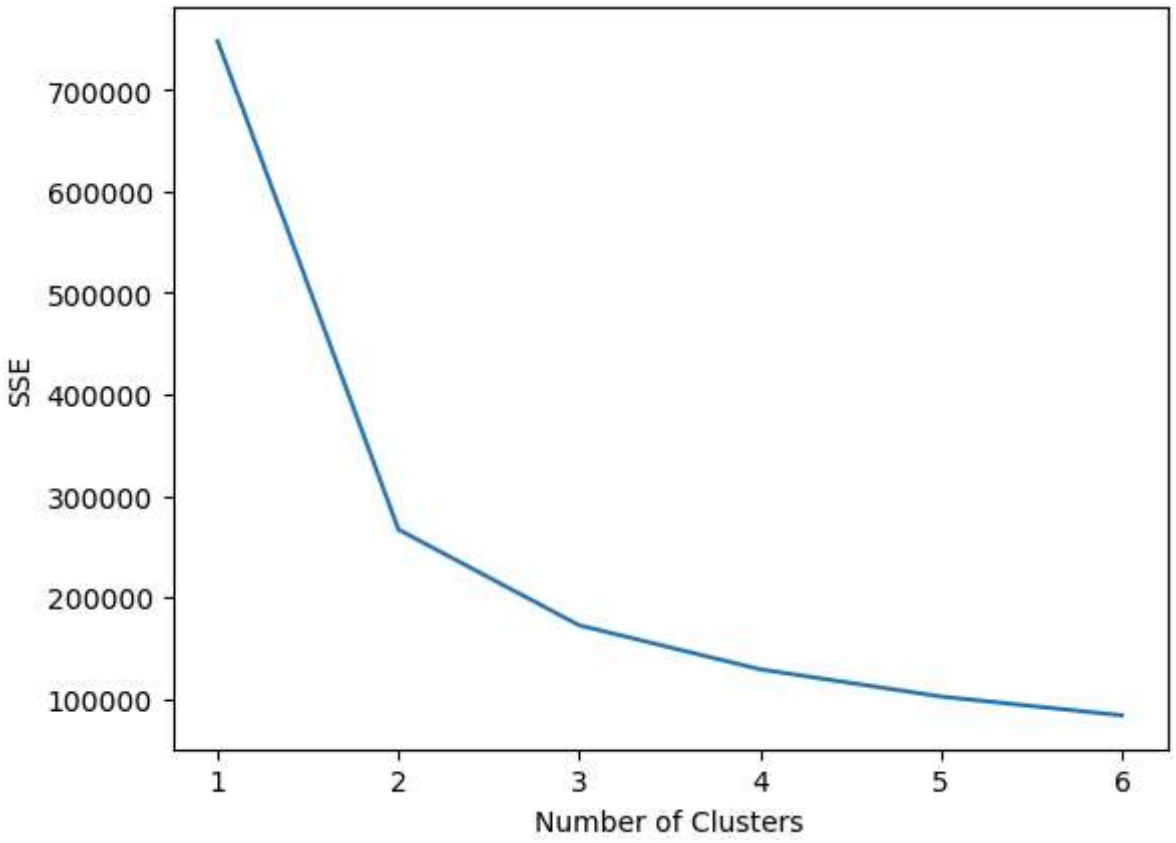
Finding Number of Clusters: Elbow Method

Using the elbow method, we can figure out what the optimal number of clusters should be. In this case, it looks like the optimal number would be either 2 or 3 clusters, since those are the points in the graph where the line begins to curve. For the rest of the code, I'll be using 3 clusters.

In [50]:
```
# Use this to get around threads warning!

import os
os.environ["OMP_NUM_THREADS"] = '2'
```

In [51]:
```
from sklearn import cluster
import matplotlib.pyplot as plt
%matplotlib inline

k_means_data = data[['imdb_rating', 'critics_score', 'audience_score']]

numClusters = [1,2,3,4,5,6]
SSE = []
for k in numClusters:
    k_means = cluster.KMeans(n_clusters=k, n_init=10)
    k_means.fit(k_means_data)
    SSE.append(k_means.inertia_)

plt.plot(numClusters, SSE)
plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
```

Out[51]:
```
Text(0, 0.5, 'SSE')
```



Training & Testing Data Split

We also need to split the data into training and testing dataframes, so we can train the K-Means model and then test it.

In [52]:
```
from sklearn.model_selection import train_test_split

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(data, data.title, test_size=0.2, r

# Recombine into dataframes
train_df = pd.DataFrame(X_train, columns=data.columns)
test_df = pd.DataFrame(X_test, columns=data.columns)


# Drop all columns except IMDB rating, critic score, and audience score
train_data = train_df[['imdb_rating', 'critics_score', 'audience_score']]
test_data = test_df[['imdb_rating', 'critics_score', 'audience_score']]
```

Training K-Means

Now that the data has been processed and split into traning and testing, we can give the training data to K-Means clustering algorithm.

In [53]:
```
# Give training data to K-Means
k_means = cluster.KMeans(n_clusters=3, max_iter=50, random_state=1, n_init = 10)
k_means.fit(train_data)
labels = k_means.labels_

pd.DataFrame(labels, index=train_df['title'], columns=['Cluster ID']).head()
```

Out[53]:

title	Cluster ID
Cold Creek Manor	1
Memento	0
Where the Heart Is	1
Amityville II: The Possession	1
Flesh And Bone	2

Centroids

Now that the K-Means model has been trained, we can view what the centroid values look like. The centroid is the mean of the data points assigned to that particular cluster. Since we have 3 clusters, we will end up with 3 centroids.

In [54]:
```
centroids = k_means.cluster_centers_
pd.DataFrame(centroids,columns=train_data.columns)
```

Out[54]:

	imdb_rating	critics_score	audience_score
0	7.374468	85.718085	80.372340
1	5.335417	20.576389	42.527778
2	6.428221	56.822086	57.803681

Testing K-Means on Testing Data

Now that we have fitted the K-Means model, we can give it testing data to see how the data will be grouped.

In [55]:
```
import numpy as np

labels = k_means.predict(test_data)
labels = labels.reshape(-1, 1)

movie_titles = np.array(test_df.title).reshape(-1, 1)

num_data = np.concatenate((movie_titles, test_data, labels), axis=1)

cols = test_data.columns.tolist() + ['Cluster ID']
predicted_data = pd.DataFrame(num_data, columns=['title'] + cols)

predicted_data.head()
```

Out[55]:

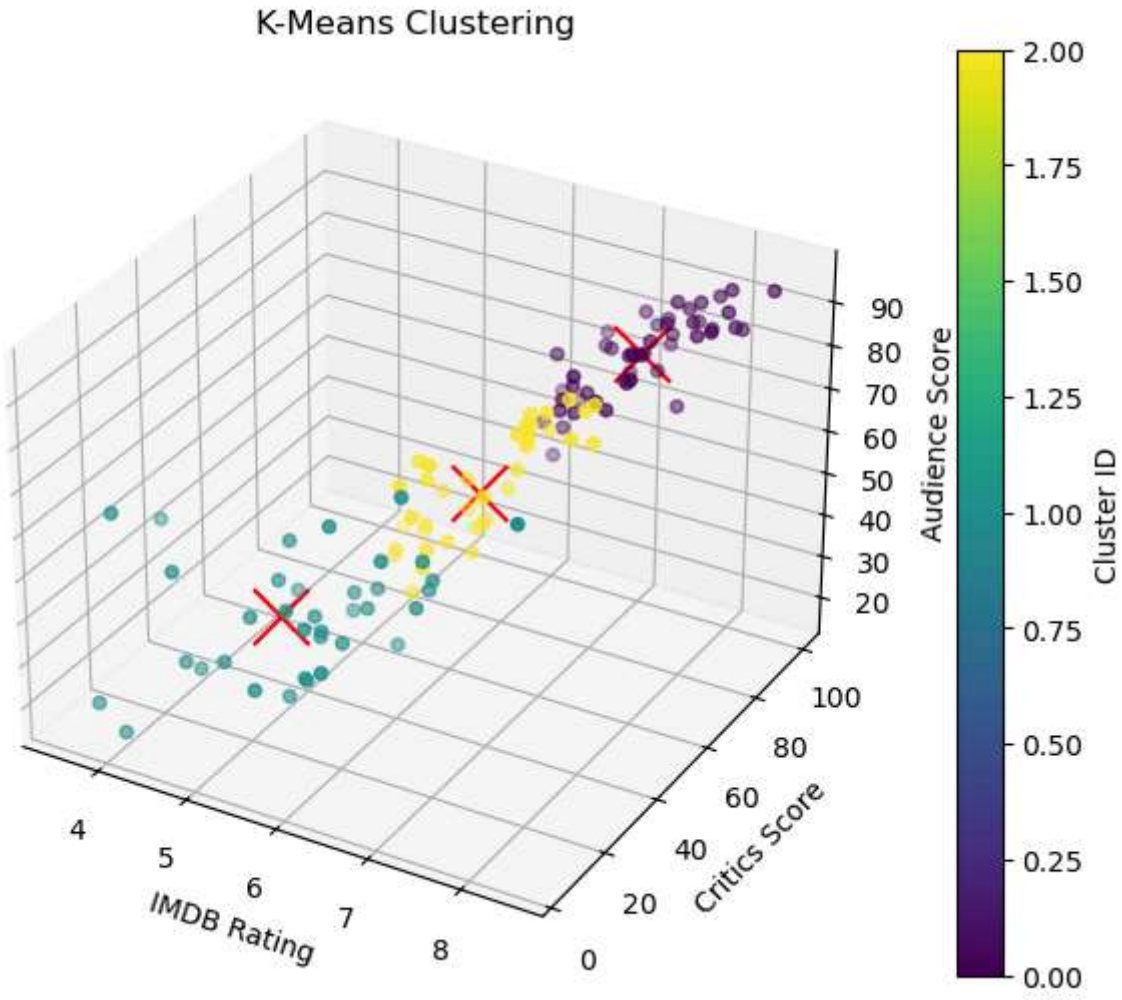
	title	imdb_rating	critics_score	audience_score	Cluster ID
0	Seven Pounds	7.7	27.0	75.0	1
1	Saving Face	7.6	87.0	88.0	0
2	Warrior	8.2	82.0	92.0	0
3	The Sure Thing	7.0	87.0	79.0	0
4	Held Up	5.1	17.0	43.0	1

Cluster Observations

We can visualize the cluster and make observations from it. K-Means may not be the best fit for this particular relationship of attributes, since the clusters are not spherical and seem to follow a more linear pattern. However, we can see that the centroids seem to be in the center of each of their respective clusters, which is to be expected.

In [56]:
```
from mpl_toolkits.mplot3d import Axes3D

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot of the data points, with each cluster assigned a different color
scatter = ax.scatter(X_test['imdb_rating'], X_test['critics_score'], X_test['audience_
ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], marker='x', s=400, c='re
```

```
# Set labels and title
ax.set_xlabel('IMDB Rating')
ax.set_ylabel('Critics Score')
ax.set_zlabel('Audience Score')
ax.set_title('K-Means Clustering')

plt.colorbar(scatter, label='Cluster ID')

plt.show()
```



Hierarchal Clustering - Penny Herrera

Single Link

In [64]:
```
from sklearn.cluster import AgglomerativeClustering

model = AgglomerativeClustering(linkage="single", distance_threshold = 7, n_clusters=N

model.fit(k_means_data)
```

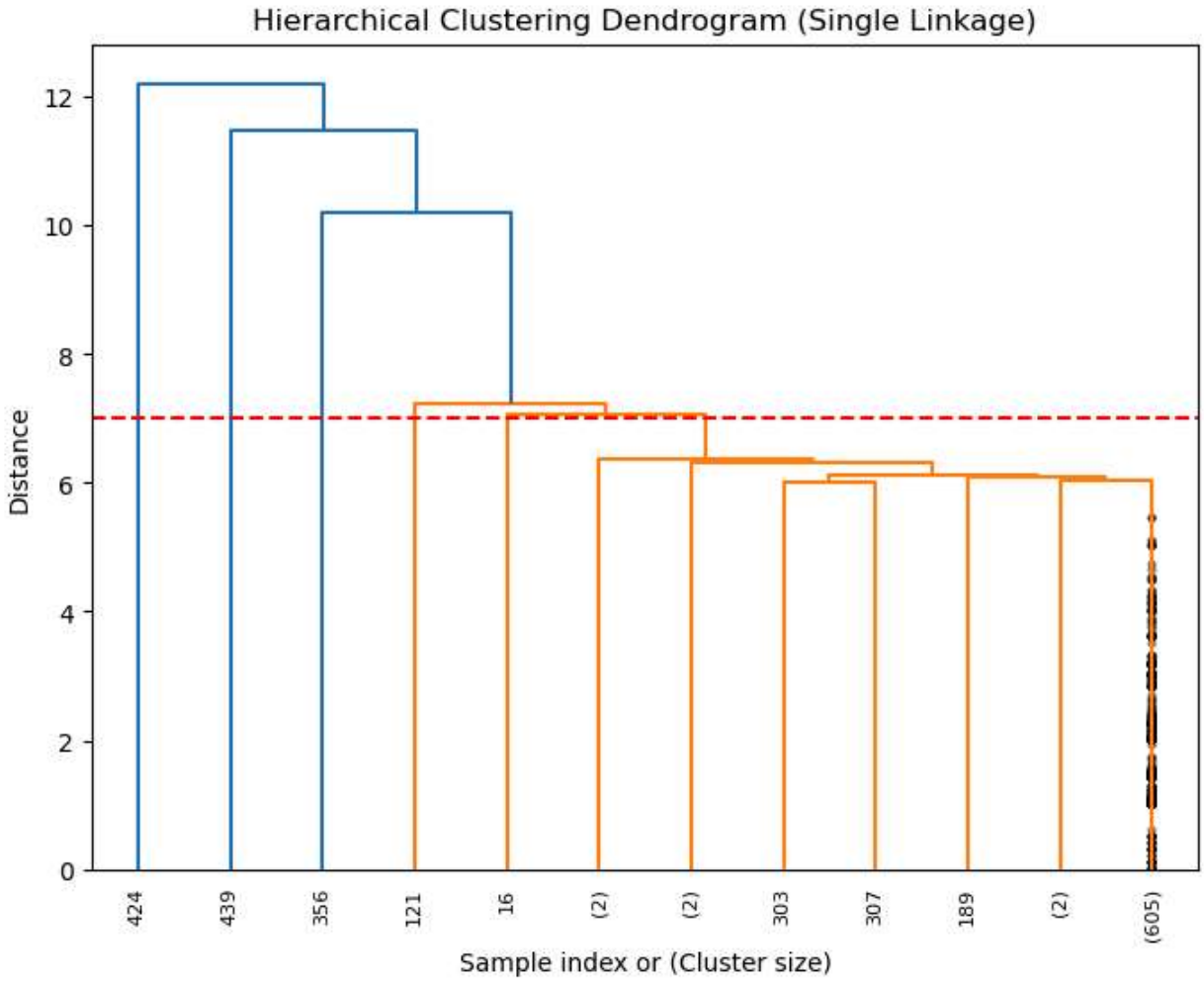
Out[64]:

AgglomerativeClustering

```
AgglomerativeClustering(distance_threshold=7, linkage='single', n_clusters=None)
```

In [65]:

```
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt

#convert DataFrame to a NumPy array
A = k_means_data.values
#generate linkage matrix using single linkage
B = linkage(A, 'single')

#plot dendrogram
plt.figure(figsize=(8, 6))
plt.title('Hierarchical Clustering Dendrogram (Single Linkage)')
plt.xlabel('Sample index or (Cluster size)')
plt.ylabel('Distance')
dendrogram(
    B,
    leaf_rotation=90.,
    leaf_font_size=8.,
    truncate_mode='lastp',
    p=12,
    show_contracted=True
)
plt.axhline(y=7, color='r', linestyle='--')
plt.show()
```



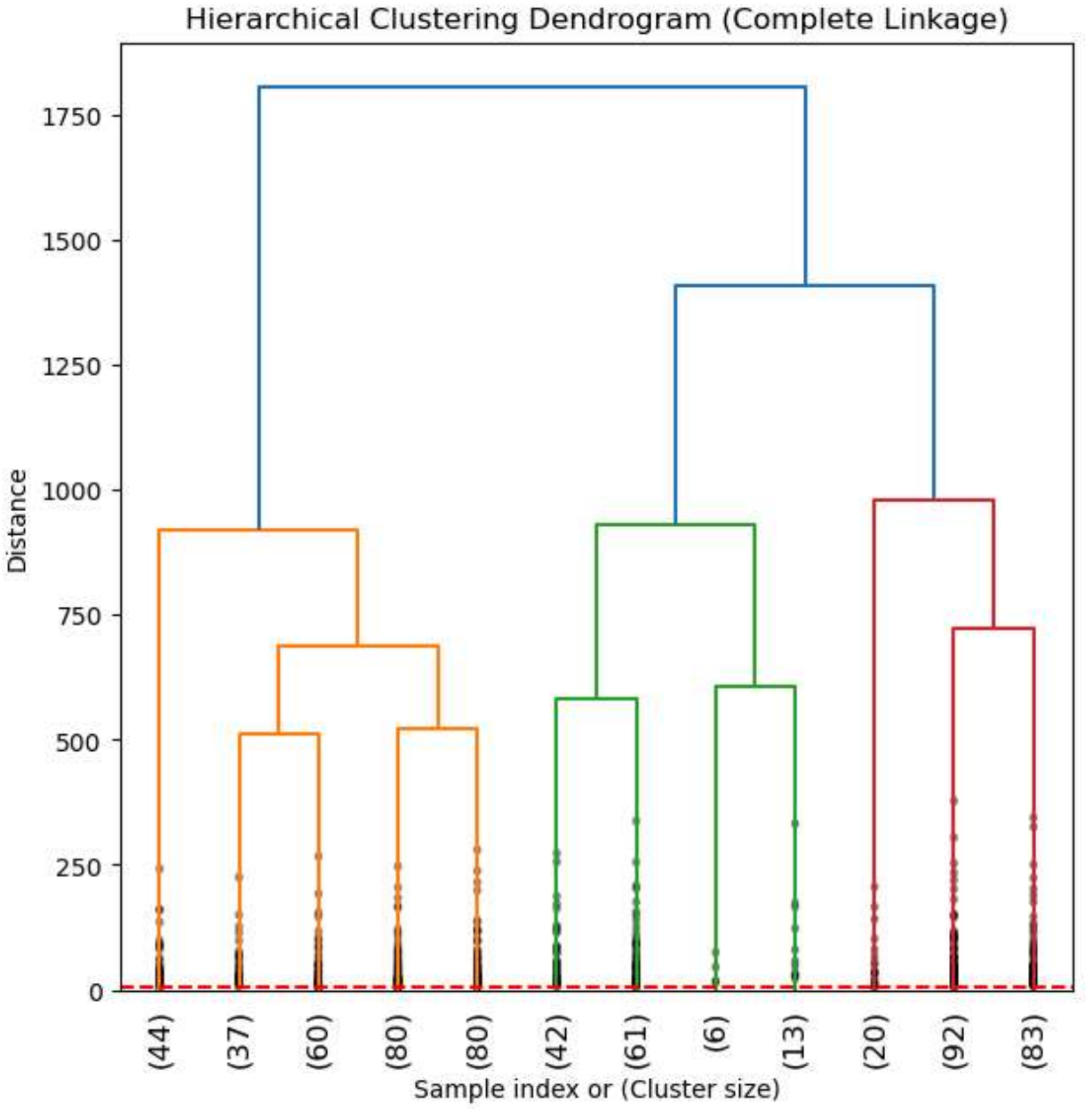
Complete Link

In [66]:
```
model = AgglomerativeClustering(linkage="complete",
                                distance_threshold = 7,
                                n_clusters=None)


model.fit(k_means_data)
```

Out[66]:
```
AgglomerativeClustering(distance_threshold=7, linkage='complete',
                        n_clusters=None)
```

In [67]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import complete, dendrogram

#computes complete linkage matrix
C = complete(B)

def complete_dendrogram(C, max_d=7, **kwargs):
    #create dendrogram
    plt.figure(figsize=(7, 7))
    plt.title("Hierarchical Clustering Dendrogram (Complete Linkage)")
```

```
    dendrogram(C, **kwargs)
    plt.axhline(y=max_d, color='r', linestyle='--')
    plt.xlabel('Sample index or (Cluster size)')
    plt.ylabel('Distance')
    plt.show()

#plot dendrogram
complete_dendrogram(C, max_d=7, truncate_mode='lastp', p=12, leaf_rotation=90, leaf_fc
```



Average Link

In [68]:
```
model = AgglomerativeClustering(linkage="average",
                                distance_threshold = 7,
                                n_clusters=None)


model.fit(k_means_data)
```

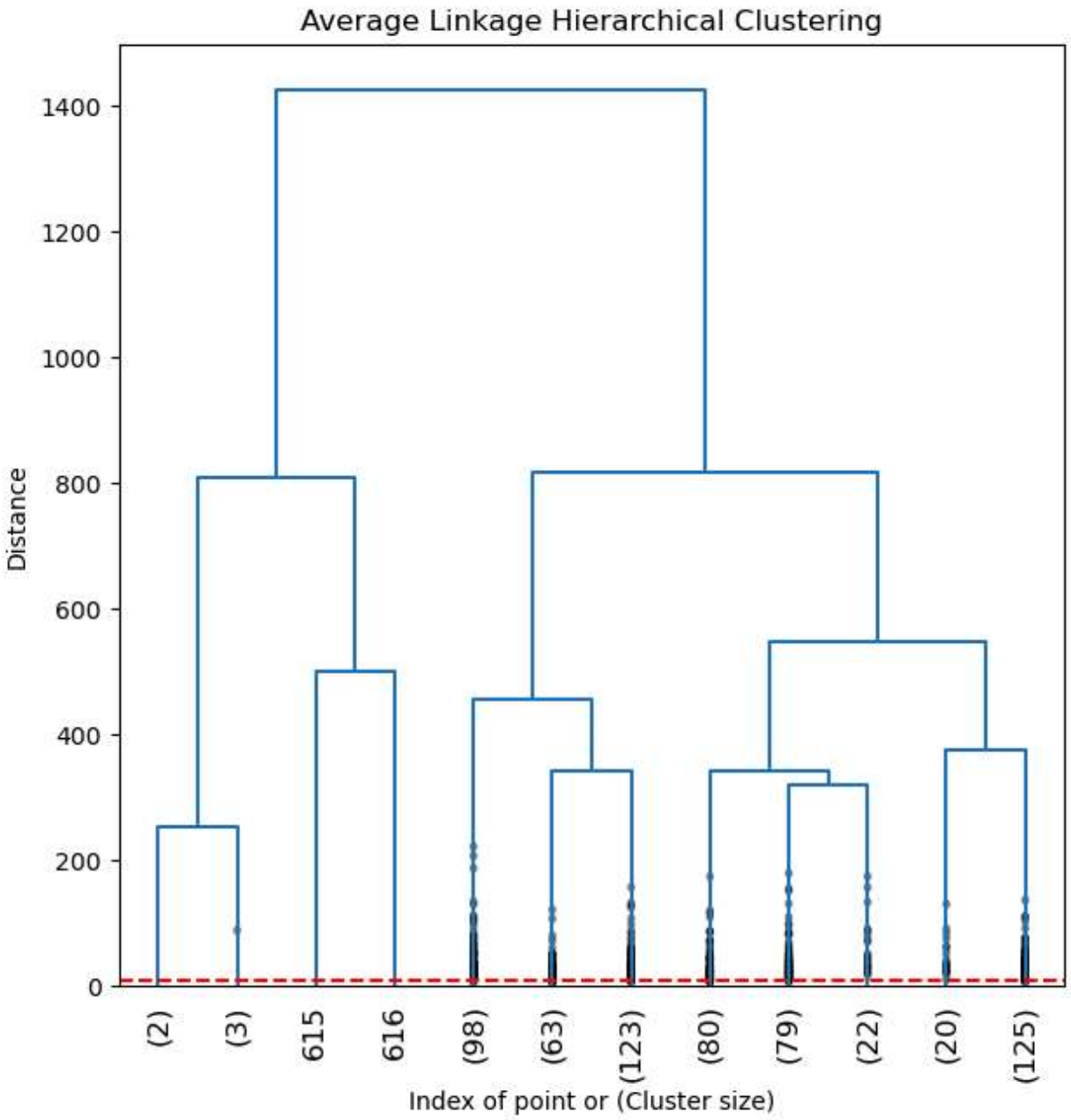
Out[68]:

AgglomerativeClustering

```
AgglomerativeClustering(distance_threshold=7, linkage='average',
                        n_clusters=None)
```

In [69]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import average, dendrogram

#computes average linkage matrix
linkage_matrix = average(C)

def average_dendrogram(linkage_matrix, max_distance=7, **kwargs):
    #create dendrogram
    plt.figure(figsize=(7, 7))
    plt.title("Average Linkage Hierarchical Clustering")
    dendrogram(
        linkage_matrix,
        color_threshold=max_distance,
        **kwargs
    )

    plt.axhline(y=max_distance, color='r', linestyle='--')
    plt.xlabel('Index of point or (Cluster size)')
    plt.ylabel('Distance')
    plt.show()

#plot & customize dendrogram
average_dendrogram(
    linkage_matrix,
    max_distance=7,
    truncate_mode='lastp',
    p=12,
    leaf_rotation=90,
    leaf_font_size=12,
    show_contracted=True
)
```


Average Linkage Hierarchical Clustering
Distance
1400
1200
1000
800
600
400
200
0
(2)
(3)
615
616
(98)
(63)
(123)
(80)
(79)
(22)
(20)
(125)
Index of point or (Cluster size)

In [ ]:

Part 2: Text Mining

Editor: Robby Dosanjh, Maddie Ananda, Warisara Lee

2.3. Your task is to create a count vector and a tfidf vector on the given data (refer to 2. In the resources below).
2.4. Display the count vector and tfidf vector and explain the usage of tfidf.

In [1]:
```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

data = [ 'Now for manners use has company believe parlors.',
         'Least nor party who wrote while did. Excuse formed as is agreed admire so on
         'Put use set uncommonly announcing and travelling. Allowance sweetness direct
         'Principle oh explained excellent do my suspected conveying in.',
         'Excellent you did therefore perfectly supposing described. ',
         'Its had resolving otherwise she contented therefore.',
         'Afford relied warmth out sir hearts sister use garden.',
         'Men day warmth formed admire former simple.',
         'Humanity declared vicinity continue supplied no an. He hastened am no proper
         'Dissimilar comparison no terminated devonshire no literature on. Say most ye

# Create count vector
count_vectorizer = CountVectorizer()
count_vector = count_vectorizer.fit_transform(data)
print("Count Vector:\n", count_vector.toarray())

# Create tfidf vector
tfidf_vectorizer = TfidfVectorizer()
tfidf_vector = tfidf_vectorizer.fit_transform(data)
print("\nTF-IDF Vector:\n", tfidf_vector.toarray())
```

```
Count Vector:
 [[0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0
  0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 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 1 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0
  0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
  0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0]
 [0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
  0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0
  0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 1 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 1 0 1 0 0 0 0 1 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 1 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
  0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1
  0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [0 1 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 1 0 0 0 0
  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
  0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 1 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
  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1
  0 0 1 0 0 0 0 0 0 0 0 0 0 2 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
  1 0 0 0 0 0 1 0 1 0 0 1 0 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
  0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0]]

TF-IDF Vector:
 [[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.36386176 0.36386176 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.36386176 0.
  0.         0.         0.         0.36386176 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.36386176 0.         0.
  0.         0.         0.         0.         0.36386176 0.
  0.         0.         0.         0.         0.         0.36386176
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.27061472 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.21513872 0.         0.25307717 0.         0.         0.
  0.         0.         0.21513872 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.21513872 0.         0.         0.         0.
  0.         0.25307717 0.         0.         0.         0.21513872
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.25307717 0.
  0.         0.25307717 0.         0.         0.         0.
  0.         0.         0.         0.25307717 0.         0.
  0.         0.21513872 0.         0.         0.25307717 0.
  0.25307717 0.         0.         0.         0.         0.
```

```
 0.         0.25307717 0.         0.         0.         0.
 0.         0.         0.         0.25307717 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.25307717
 0.25307717 0.25307717 0.         0.        ]
[0.         0.         0.         0.28541401 0.         0.
 0.28541401 0.28541401 0.24262799 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.28541401 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.         0.28541401 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.28541401 0.
 0.         0.         0.         0.         0.28541401 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.28541401 0.         0.         0.28541401
 0.28541401 0.28541401 0.21227082 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.33859118 0.         0.         0.
 0.         0.         0.         0.         0.33859118 0.
 0.28783344 0.         0.         0.33859118 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.33859118 0.         0.
 0.         0.         0.         0.         0.         0.
 0.33859118 0.         0.         0.         0.         0.
 0.33859118 0.         0.         0.         0.         0.
 0.         0.         0.33859118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.33859118 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.         0.40265122
 0.         0.34229032 0.         0.         0.         0.
 0.34229032 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.         0.40265122 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.40265122 0.         0.         0.         0.34229032 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.40265122]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.38568218 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.38568218 0.         0.         0.
 0.         0.         0.         0.         0.         0.38568218
```

```
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.38568218 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.38568218 0.         0.         0.         0.         0.38568218
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.32786509 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.34761243 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.         0.34761243 0.         0.         0.         0.
 0.         0.34761243 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.34761243 0.         0.
 0.         0.         0.         0.         0.         0.34761243
 0.         0.         0.         0.         0.         0.
 0.         0.34761243 0.34761243 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.25852961 0.         0.29550232 0.
 0.         0.         0.         0.        ]
[0.34229032 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.40265122 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.34229032
 0.40265122 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.40265122 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.
 0.40265122 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.34229032 0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.25914545 0.25914545
 0.         0.         0.         0.         0.         0.
 0.         0.25914545 0.         0.         0.25914545 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.25914545 0.         0.         0.
 0.         0.         0.         0.         0.25914545 0.25914545
 0.         0.         0.25914545 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.44059462 0.         0.         0.25914545
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.25914545 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.25914545
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.25914545 0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.24534225
 0.         0.         0.         0.         0.         0.
```

```
0.24534225 0.         0.         0.24534225 0.         0.24534225
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.24534225 0.         0.         0.         0.         0.
0.24534225 0.         0.24534225 0.         0.         0.24534225
0.         0.         0.41712666 0.         0.         0.
0.         0.20856333 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.24534225 0.24534225 0.         0.
0.         0.         0.         0.         0.24534225 0.
0.         0.         0.         0.24534225 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.24534225 0.        ]]
```

The usage of TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a weighting scheme that accounts for how important a word is in a document corpus. It's a way to score the importance of words (or terms) in a document based on how frequently they appear across multiple documents. If a word appears frequently in a document, but not in many documents, it is considered to be important. It consists of two parts:

Term Frequency (TF): The count of how many times the word occurs in the document. Common words will have a high TF.

Inverse Document Frequency (IDF): A downscaling factor based on how many documents in the corpus contain the word. Words appearing in many documents have a lower IDF.

By multiplying TF and IDF, we get the TF-IDF weight. It allows highlighting words that are distinct or characteristic for a particular document, while downplaying words that are very common. TF-IDF vectors are commonly used as features for machine learning on text, as they better represent the importance of words than just frequencies.

Cluster Analysis, ANN, & Text Mining: Part 3 - ANN

Author: Michael Berbach, Isaiah Samaniego

In [115…
```
import pandas as pd
from keras import Sequential
from keras.layers import Dense
import numpy as np
from sklearn import preprocessing as prep
from sklearn.preprocessing import OneHotEncoder
import tensorflow as ff
from sklearn.metrics import accuracy_score, precision_score, classification_report, f1
```

We import and parse the Admissions dataset, and then we display it.

In [116…
```
# loading data set from assignment 2
df = pd.read_csv('data\Admission_Predict_Ver1.1_small_data_set_for_Linear_Regression.c
df = df.drop(columns=['Serial No.'])
print(df)
```

```
     GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research  \
0          337          118                  4  4.5  4.5  9.65         1
1          324          107                  4  4.0  4.5  8.87         1
2          316          104                  3  3.0  3.5  8.00         1
3          322          110                  3  3.5  2.5  8.67         1
4          314          103                  2  2.0  3.0  8.21         0
..         ...          ...                ...  ...  ...   ...       ...
495        332          108                  5  4.5  4.0  9.02         1
496        337          117                  5  5.0  5.0  9.87         1
497        330          120                  5  4.5  5.0  9.56         1
498        312          103                  4  4.0  5.0  8.43         0
499        327          113                  4  4.5  4.5  9.04         0

     Chance of Admit
0               0.92
1               0.76
2               0.72
3               0.80
4               0.65
..               ...
495             0.87
496             0.96
497             0.93
498             0.73
499             0.84

[500 rows x 8 columns]
```

```
C:\Users\legen\AppData\Local\Temp\ipykernel_23004\3656229374.py:2: ParserWarning: Fal
ling back to the 'python' engine because the 'c' engine does not support regex separa
tors (separators > 1 char and different from '\s+' are interpreted as regex); you can
avoid this warning by specifying engine='python'.
  df = pd.read_csv('data\Admission_Predict_Ver1.1_small_data_set_for_Linear_Regressio
n.csv', sep=r'\s*,\s*')
```

Preprocessing ANN: Normalize Predictors and Binarize Targets

Using the min-max approach, we rescale to the range of [0,1]. This is done by subtracting the minimum value and dividing by the range.

In [117…
```
df['GRE Score'] = ((df['GRE Score'] - df['GRE Score'].min()) / (df['GRE Score'].max()
df['TOEFL Score'] = ((df['TOEFL Score'] - df['TOEFL Score'].min()) / (df['TOEFL Score'
df['University Rating'] = ((df['University Rating'] - df['University Rating'].min()) /
df['SOP'] = ((df['SOP'] - df['SOP'].min()) / (df['SOP'].max() - df['SOP'].min())).asty
df['CGPA'] = ((df['CGPA'] - df['CGPA'].min()) / (df['CGPA'].max() - df['CGPA'].min()))
df['Research'] = ((df['Research'] - df['Research'].min()) / (df['Research'].max() - df
df['LOR'] = ((df['LOR'] - df['LOR'].min()) / (df['LOR'].max() - df['LOR'].min())).asty
df['Chance of Admit'] = (df['Chance of Admit'] > df['Chance of Admit'].median()).astyp
print(df)
```

```
     GRE Score  TOEFL Score  University Rating    SOP    LOR      CGPA  \
0         0.94     0.928571               0.75  0.875  0.875  0.913462
1         0.68     0.535714               0.75  0.750  0.875  0.663462
2         0.52     0.428571               0.50  0.500  0.625  0.384615
3         0.64     0.642857               0.50  0.625  0.375  0.599359
4         0.48     0.392857               0.25  0.250  0.500  0.451923
..         ...          ...                ...    ...    ...       ...
495       0.84     0.571429               1.00  0.875  0.750  0.711538
496       0.94     0.892857               1.00  1.000  1.000  0.983974
497       0.80     1.000000               1.00  0.875  1.000  0.884615
498       0.44     0.392857               0.75  0.750  1.000  0.522436
499       0.74     0.750000               0.75  0.875  0.875  0.717949

     Research  Chance of Admit
0           1                1
1           1                1
2           1                0
3           1                1
4           0                0
..        ...              ...
495         1                1
496         1                1
497         1                1
498         0                1
499         0                1

[500 rows x 8 columns]
```

We will now create a test set that will take 100 rows from our dataset.

In [118…
```
index = 400
test = df[index:]
train = df[:index]
test.head(20)
```

Out[118]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
400	0.28	0.285714	0.25	0.625	0.500	0.455128	0	0
401	0.50	0.464286	0.25	0.500	0.500	0.493590	0	0
402	0.68	0.607143	0.50	0.625	0.500	0.685897	1	1
403	0.80	0.857143	0.75	0.750	0.625	0.778846	1	1
404	0.42	0.321429	0.50	0.250	0.375	0.269231	1	0
405	0.24	0.250000	0.50	0.375	0.500	0.208333	0	0
406	0.64	0.392857	0.75	0.500	0.375	0.391026	1	0
407	0.16	0.285714	0.50	0.375	0.750	0.368590	1	0
408	0.14	0.321429	0.50	0.250	0.750	0.278846	1	0
409	0.20	0.214286	0.00	0.250	0.375	0.391026	0	0
410	0.22	0.142857	0.00	0.500	0.750	0.243590	0	0
411	0.46	0.071429	0.25	0.375	0.125	0.426282	0	0
412	0.48	0.357143	0.75	0.375	0.250	0.346154	1	0
413	0.54	0.321429	0.50	0.500	0.250	0.365385	1	0
414	0.62	0.642857	0.75	0.625	0.750	0.496795	1	0
415	0.74	0.500000	0.75	0.750	0.875	0.625000	1	1
416	0.50	0.428571	0.50	0.750	0.375	0.416667	0	0
417	0.52	0.392857	0.50	0.625	0.250	0.282051	0	0
418	0.38	0.678571	0.25	0.375	0.750	0.394231	0	0
419	0.36	0.357143	0.25	0.250	0.625	0.378205	1	0

In [119…

```
import collections
def to_xy(df, target):
    result = []
    for x in df.columns:
        if x != target:
            result.append(x)
    # find out the type of the target column.
    target_type = df[target].dtypes
    target_type = target_type[0] if isinstance(target_type, collections.abc.Sequence)
    # Encode to int for classification, float otherwise. TensorFlow likes 32 bits.
    if target_type in (np.int64, np.int32):
        # Classification
        dummies = pd.get_dummies(df[target])
        return df[result].values.astype(np.float32), dummies.values.astype(np.float32)
    else:
        # Regression
        return df[result].values.astype(np.float32), df[target].values.astype(np.float

X, Y = to_xy(train,'Chance of Admit')
testX, testY = to_xy(test, 'Chance of Admit')
```

In [120…

```
print(X.shape)
print(Y.shape)
Y
```

```
(400, 7)
(400, 2)
```

Out[120]:
```
array([[0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
```

```
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
```

```
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
```

```
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
```

```
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
```

```
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
```

```
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.]], dtype=float32)
```

In [121…

```
model = ff.keras.Sequential()
model.add(Dense(12, input_dim = X.shape[1], activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(2, activation='softmax'))
```

```
C:\Users\legen\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWar
ning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequen
tial models, prefer using an `Input(shape)` object as the first layer in the model in
stead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [122…

```
model.compile(loss='categorical_crossentropy',optimizer='adam')
model.fit(X,Y,verbose=2, epochs=100)
```

```
Epoch 1/100
13/13 - 1s - 51ms/step - loss: 0.7166
Epoch 2/100
13/13 - 0s - 2ms/step - loss: 0.6957
Epoch 3/100
13/13 - 0s - 2ms/step - loss: 0.6803
Epoch 4/100
13/13 - 0s - 2ms/step - loss: 0.6706
Epoch 5/100
13/13 - 0s - 2ms/step - loss: 0.6626
Epoch 6/100
13/13 - 0s - 2ms/step - loss: 0.6527
Epoch 7/100
13/13 - 0s - 2ms/step - loss: 0.6411
Epoch 8/100
13/13 - 0s - 2ms/step - loss: 0.6279
Epoch 9/100
13/13 - 0s - 2ms/step - loss: 0.6133
Epoch 10/100
13/13 - 0s - 2ms/step - loss: 0.5987
Epoch 11/100
13/13 - 0s - 2ms/step - loss: 0.5829
Epoch 12/100
13/13 - 0s - 2ms/step - loss: 0.5659
Epoch 13/100
13/13 - 0s - 2ms/step - loss: 0.5480
Epoch 14/100
13/13 - 0s - 2ms/step - loss: 0.5315
Epoch 15/100
13/13 - 0s - 2ms/step - loss: 0.5146
Epoch 16/100
13/13 - 0s - 2ms/step - loss: 0.4997
Epoch 17/100
13/13 - 0s - 2ms/step - loss: 0.4850
Epoch 18/100
13/13 - 0s - 2ms/step - loss: 0.4715
Epoch 19/100
13/13 - 0s - 2ms/step - loss: 0.4595
Epoch 20/100
13/13 - 0s - 2ms/step - loss: 0.4480
Epoch 21/100
13/13 - 0s - 2ms/step - loss: 0.4371
Epoch 22/100
13/13 - 0s - 2ms/step - loss: 0.4289
Epoch 23/100
13/13 - 0s - 2ms/step - loss: 0.4184
Epoch 24/100
13/13 - 0s - 2ms/step - loss: 0.4109
Epoch 25/100
13/13 - 0s - 2ms/step - loss: 0.4026
Epoch 26/100
13/13 - 0s - 2ms/step - loss: 0.3969
Epoch 27/100
13/13 - 0s - 2ms/step - loss: 0.3890
Epoch 28/100
13/13 - 0s - 2ms/step - loss: 0.3830
Epoch 29/100
13/13 - 0s - 2ms/step - loss: 0.3776
Epoch 30/100
13/13 - 0s - 2ms/step - loss: 0.3721
```

```
Epoch 31/100
13/13 - 0s - 2ms/step - loss: 0.3681
Epoch 32/100
13/13 - 0s - 2ms/step - loss: 0.3640
Epoch 33/100
13/13 - 0s - 2ms/step - loss: 0.3586
Epoch 34/100
13/13 - 0s - 2ms/step - loss: 0.3547
Epoch 35/100
13/13 - 0s - 2ms/step - loss: 0.3520
Epoch 36/100
13/13 - 0s - 2ms/step - loss: 0.3487
Epoch 37/100
13/13 - 0s - 2ms/step - loss: 0.3455
Epoch 38/100
13/13 - 0s - 2ms/step - loss: 0.3417
Epoch 39/100
13/13 - 0s - 2ms/step - loss: 0.3390
Epoch 40/100
13/13 - 0s - 2ms/step - loss: 0.3368
Epoch 41/100
13/13 - 0s - 2ms/step - loss: 0.3340
Epoch 42/100
13/13 - 0s - 2ms/step - loss: 0.3328
Epoch 43/100
13/13 - 0s - 2ms/step - loss: 0.3305
Epoch 44/100
13/13 - 0s - 2ms/step - loss: 0.3285
Epoch 45/100
13/13 - 0s - 2ms/step - loss: 0.3282
Epoch 46/100
13/13 - 0s - 2ms/step - loss: 0.3253
Epoch 47/100
13/13 - 0s - 2ms/step - loss: 0.3235
Epoch 48/100
13/13 - 0s - 2ms/step - loss: 0.3227
Epoch 49/100
13/13 - 0s - 2ms/step - loss: 0.3209
Epoch 50/100
13/13 - 0s - 2ms/step - loss: 0.3183
Epoch 51/100
13/13 - 0s - 2ms/step - loss: 0.3175
Epoch 52/100
13/13 - 0s - 2ms/step - loss: 0.3169
Epoch 53/100
13/13 - 0s - 2ms/step - loss: 0.3177
Epoch 54/100
13/13 - 0s - 2ms/step - loss: 0.3141
Epoch 55/100
13/13 - 0s - 2ms/step - loss: 0.3134
Epoch 56/100
13/13 - 0s - 2ms/step - loss: 0.3117
Epoch 57/100
13/13 - 0s - 2ms/step - loss: 0.3118
Epoch 58/100
13/13 - 0s - 2ms/step - loss: 0.3111
Epoch 59/100
13/13 - 0s - 2ms/step - loss: 0.3089
Epoch 60/100
13/13 - 0s - 2ms/step - loss: 0.3084
```

```
Epoch 61/100
13/13 - 0s - 2ms/step - loss: 0.3090
Epoch 62/100
13/13 - 0s - 2ms/step - loss: 0.3068
Epoch 63/100
13/13 - 0s - 2ms/step - loss: 0.3061
Epoch 64/100
13/13 - 0s - 2ms/step - loss: 0.3083
Epoch 65/100
13/13 - 0s - 2ms/step - loss: 0.3045
Epoch 66/100
13/13 - 0s - 2ms/step - loss: 0.3045
Epoch 67/100
13/13 - 0s - 2ms/step - loss: 0.3030
Epoch 68/100
13/13 - 0s - 2ms/step - loss: 0.3038
Epoch 69/100
13/13 - 0s - 2ms/step - loss: 0.3009
Epoch 70/100
13/13 - 0s - 2ms/step - loss: 0.3004
Epoch 71/100
13/13 - 0s - 2ms/step - loss: 0.2991
Epoch 72/100
13/13 - 0s - 2ms/step - loss: 0.2998
Epoch 73/100
13/13 - 0s - 2ms/step - loss: 0.2975
Epoch 74/100
13/13 - 0s - 2ms/step - loss: 0.2974
Epoch 75/100
13/13 - 0s - 2ms/step - loss: 0.3019
Epoch 76/100
13/13 - 0s - 2ms/step - loss: 0.2951
Epoch 77/100
13/13 - 0s - 2ms/step - loss: 0.2972
Epoch 78/100
13/13 - 0s - 2ms/step - loss: 0.2944
Epoch 79/100
13/13 - 0s - 2ms/step - loss: 0.2947
Epoch 80/100
13/13 - 0s - 2ms/step - loss: 0.2951
Epoch 81/100
13/13 - 0s - 2ms/step - loss: 0.2927
Epoch 82/100
13/13 - 0s - 2ms/step - loss: 0.2937
Epoch 83/100
13/13 - 0s - 2ms/step - loss: 0.2930
Epoch 84/100
13/13 - 0s - 2ms/step - loss: 0.2946
Epoch 85/100
13/13 - 0s - 2ms/step - loss: 0.2950
Epoch 86/100
13/13 - 0s - 2ms/step - loss: 0.2938
Epoch 87/100
13/13 - 0s - 2ms/step - loss: 0.2900
Epoch 88/100
13/13 - 0s - 2ms/step - loss: 0.2906
Epoch 89/100
13/13 - 0s - 2ms/step - loss: 0.2905
Epoch 90/100
13/13 - 0s - 2ms/step - loss: 0.2897
```

```
Epoch 91/100
13/13 - 0s - 2ms/step - loss: 0.2893
Epoch 92/100
13/13 - 0s - 2ms/step - loss: 0.2883
Epoch 93/100
13/13 - 0s - 2ms/step - loss: 0.2887
Epoch 94/100
13/13 - 0s - 2ms/step - loss: 0.2882
Epoch 95/100
13/13 - 0s - 2ms/step - loss: 0.2878
Epoch 96/100
13/13 - 0s - 2ms/step - loss: 0.2905
Epoch 97/100
13/13 - 0s - 2ms/step - loss: 0.2868
Epoch 98/100
13/13 - 0s - 2ms/step - loss: 0.2893
Epoch 99/100
13/13 - 0s - 2ms/step - loss: 0.2890
Epoch 100/100
13/13 - 0s - 2ms/step - loss: 0.2866
```

Out[122]:
```
<keras.src.callbacks.history.History at 0x2003f151950>
```

In [123…
```
pred = model.predict(testX)
print(pred[0])
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
[0.9360205  0.06397951]
```

In [124…
```
pred = np.argmax(pred, axis=1)
```

In [125…
```
true = np.argmax(testY, axis=1)
```

In [126…
```
print("Predicted: ", [pred])
print("True: ", [true])
```

```
Predicted:  [array([0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1], dtype=int64)]
True:  [array([0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1], dtype=int64)]
```

We generate accuracy from the ANN and classification report, we can see that our predicted values are above average.

In [127…
```
print('Accuracy on test data is %.2f' % (accuracy_score(true, pred)))
```

```
Accuracy on test data is 0.90
```

In [128…
```
print(classification_report(true,pred))
```

	precision	recall	f1-score	support
0	0.91	0.91	0.91	56
1	0.89	0.89	0.89	44
accuracy			0.90	100
macro avg	0.90	0.90	0.90	100
weighted avg	0.90	0.90	0.90	100

In [ ]: