# cluster_analysis

December 1, 2023

## 1 Part 1: Cluster Analysis

Installation to install on python/anaconda `pip install tensorflow` `pip install seaborn`

This code block is made more as a setup with giving all the necessary imports and functions to use for Cluster Analysis.

[ ]:
```
import os
import numpy as np
import pandas as pd
import collections
import matplotlib.pyplot as plt
import shutil
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
↪recall_score, classification_report, ConfusionMatrixDisplay, confusion_matrix
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split # Import train_test_split␣
↪function
from sklearn import metrics #Import scikit-learn metrics module for accuracy␣
↪calculation
from sklearn import preprocessing, cluster
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler

%matplotlib inline
plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams['figure.dpi'] = 100

# Encode text values to indexes(i.e. [1],[2],[3] for red,green,blue).
def encode_text_index(data, name):
    le = preprocessing.LabelEncoder()
    data[name] = le.fit_transform(data[name])
    return le.classes_

# Create dummies columns from categorical values
def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name], prefix=name)
    df = pd.concat([df, dummies], axis=1)
```

```
    df.drop(name, axis=1, inplace=True)
    return df
```

## 1.1 KMeans Implementation

Below we take the imdb dataset and print it while fixing a small issue with the unnamed numbered column.

[ ]:
```
imdb_dataset = pd.read_csv("./imdb_dataset.csv")
# Seems the csv file is missing the 1st column name
imdb_dataset.rename(columns={'Unnamed: 0':'id'}, inplace=True)
print("All columns: ", imdb_dataset.columns)
imdb_dataset.head(10)
```

```
All columns:  Index(['id', 'title', 'title_type', 'genre', 'runtime',
'mpaa_rating',
       'studio', 'thtr_rel_year', 'thtr_rel_month', 'thtr_rel_day',
       'dvd_rel_year', 'dvd_rel_month', 'dvd_rel_day', 'imdb_rating',
       'imdb_num_votes', 'critics_rating', 'critics_score', 'audience_rating',
       'audience_score', 'best_pic_nom', 'best_pic_win', 'best_actor_win',
       'best_actress_win', 'best_dir_win', 'top200_box', 'director', 'actor1',
       'actor2', 'actor3', 'actor4', 'actor5', 'imdb_url', 'rt_url'],
      dtype='object')
```

[ ]:

| | id | title | title_type | genre | runtime | mpaa_rating | \ |
|---|---|---|---|---|---|---|---|
| 0 | 1 | Filly Brown | Feature Film | Drama | 80.0 | R | |
| 1 | 2 | The Dish | Feature Film | Drama | 101.0 | PG-13 | |
| 2 | 3 | Waiting for Guffman | Feature Film | Comedy | 84.0 | R | |
| 3 | 4 | The Age of Innocence | Feature Film | Drama | 139.0 | PG | |
| 4 | 5 | Malevolence | Feature Film | Horror | 90.0 | R | |
| 5 | 6 | Old Partner | Documentary | Documentary | 78.0 | Unrated | |
| 6 | 7 | Lady Jane | Feature Film | Drama | 142.0 | PG-13 | |
| 7 | 8 | Mad Dog Time | Feature Film | Drama | 93.0 | R | |
| 8 | 9 | Beauty Is Embarrassing | Documentary | Documentary | 88.0 | Unrated | |
| 9 | 10 | The Snowtown Murders | Feature Film | Drama | 119.0 | Unrated | |

| | studio | thtr_rel_year | thtr_rel_month | thtr_rel_day | … | \ |
|---|---|---|---|---|---|---|
| 0 | Indomina Media Inc. | 2013 | 4 | 19 | … | |
| 1 | Warner Bros. Pictures | 2001 | 3 | 14 | … | |
| 2 | Sony Pictures Classics | 1996 | 8 | 21 | … | |
| 3 | Columbia Pictures | 1993 | 10 | 1 | … | |
| 4 | Anchor Bay Entertainment | 2004 | 9 | 10 | … | |
| 5 | Shcalo Media Group | 2009 | 1 | 15 | … | |
| 6 | Paramount Home Video | 1986 | 1 | 1 | … | |
| 7 | MGM/United Artists | 1996 | 11 | 8 | … | |
| 8 | Independent Pictures | 2012 | 9 | 7 | … | |
| 9 | IFC Films | 2012 | 3 | 2 | … | |

| | best_dir_win | top200_box | director | actor1 |
|---|---|---|---|---|
| 0 | no | no | Michael D. Olmos | Gina Rodriguez |
| 1 | no | no | Rob Sitch | Sam Neill |
| 2 | no | no | Christopher Guest | Christopher Guest |
| 3 | yes | no | Martin Scorsese | Daniel Day-Lewis |
| 4 | no | no | Stevan Mena | Samantha Dark |
| 5 | no | no | Chung-ryoul Lee | Choi Won-kyun |
| 6 | no | no | Trevor Nunn | Cary Elwes |
| 7 | no | no | Larry Bishop | Richard Dreyfuss |
| 8 | no | no | Neil Berkeley | Paul Reubens |
| 9 | no | no | Justin Kurzel | Lucas Pittaway |

| | actor2 | actor3 | actor4 |
|---|---|---|---|
| 0 | Jenni Rivera | Lou Diamond Phillips | Emilio Rivera |
| 1 | Kevin Harrington | Patrick Warburton | Tom Long |
| 2 | Catherine O'Hara | Parker Posey | Eugene Levy |
| 3 | Michelle Pfeiffer | Winona Ryder | Richard E. Grant |
| 4 | R. Brandon Johnson | Brandon Johnson | Heather Magee |
| 5 | Lee Sam-soon | Moo | NaN |
| 6 | John Wood | Michael Hordern | Jill Bennett II |
| 7 | Jeff Goldblum | Gabriel Byrne | Ellen Barkin |
| 8 | Matt Groening | Todd Oldham | Jonathan Dayton |
| 9 | Daniel Henshall | Louise Harris | Craig Coyne |

| | actor5 | imdb_url |
|---|---|---|
| 0 | Joseph Julian Soria | http://www.imdb.com/title/tt1869425/ |
| 1 | Genevieve Mooy | http://www.imdb.com/title/tt0205873/ |
| 2 | Bob Balaban | http://www.imdb.com/title/tt0118111/ |
| 3 | Alec McCowen | http://www.imdb.com/title/tt0106226/ |
| 4 | Richard Glover | http://www.imdb.com/title/tt0388230/ |
| 5 | NaN | http://www.imdb.com/title/tt1334549/ |
| 6 | Helena Bonham Carter | http://www.imdb.com/title/tt0091374/ |
| 7 | Diane Lane | http://www.imdb.com/title/tt0116953/ |
| 8 | Cliff Benjamin | http://www.imdb.com/title/tt2040281/ |
| 9 | Richard Green | http://www.imdb.com/title/tt1680114/ |

| | rt_url |
|---|---|
| 0 | //www.rottentomatoes.com/m/filly_brown_2012/ |
| 1 | //www.rottentomatoes.com/m/dish/ |
| 2 | //www.rottentomatoes.com/m/waiting_for_guffman/ |
| 3 | //www.rottentomatoes.com/m/age_of_innocence/ |
| 4 | //www.rottentomatoes.com/m/10004684-malevolence/ |
| 5 | //www.rottentomatoes.com/m/old-partner/ |
| 6 | //www.rottentomatoes.com/m/lady_jane/ |
| 7 | //www.rottentomatoes.com/m/mad_dog_time/ |
| 8 | //www.rottentomatoes.com/m/beauty_is_embarrass… |
| 9 | //www.rottentomatoes.com/m/the_snowtown_murders/ |

```
[10 rows x 33 columns]
```

### 1.1.1 Vertical Partitioning

Select features for K-means clustering. Because of the large size of data, we first need to partition it in order to prepare us for clustering analysis. We first do vertical partitioning in order to isolate the necessary columns we will use.

[ ]:
```
db = imdb_dataset[['title', 'genre', 'mpaa_rating', 'imdb_rating',␣
↪'critics_score', 'audience_rating','audience_score']]
db
```

[ ]:
```
                           title               genre mpaa_rating  imdb_rating  \
0                    Filly Brown               Drama           R          5.5
1                       The Dish               Drama       PG-13          7.3
2            Waiting for Guffman              Comedy           R          7.6
3           The Age of Innocence               Drama          PG          7.2
4                    Malevolence              Horror           R          5.1
..                           …                   …         …           …
646            Death Defying Acts               Drama          PG          5.9
647                   Half Baked              Comedy           R          6.7
648             Dance of the Dead  Action & Adventure           R          5.9
649  Around the World in 80 Days  Action & Adventure          PG          5.8
650                          LOL              Comedy       PG-13          4.2

     critics_score audience_rating  audience_score
0               45         Upright              73
1               96         Upright              81
2               91         Upright              91
3               80         Upright              76
4               33         Spilled              27
..             …             …            …
646             44         Spilled              26
647             29         Upright              81
648             80         Spilled              52
649             31         Spilled              34
650             17         Spilled              51

[651 rows x 7 columns]
```

### 1.1.2 Preprocess categorical columns and normalize numerical columns

Note: we drop 'title' as is not informative for K-means clustering and 'genre' because we want to use 'genre' later to analyze our clusters

The next step for preprocessing we use is hot encoding in order to binarize the db2 values into 0s to 1s using the z-score normalization.

[ ]:
```
db2 = encode_text_dummy(db, 'mpaa_rating')
db2 = encode_text_dummy(db2, 'audience_rating')
db2_preprocessed = db2.drop(columns=['title', 'genre'])
# # Z-score each column
db2_preprocessed = (db2_preprocessed-db2_preprocessed.mean()) /␣
↪db2_preprocessed.std()
db2_preprocessed
```

[ ]:
```
     imdb_rating  critics_score  audience_score  mpaa_rating_G  \
0      -0.915502      -0.446720        0.526019      -0.173254
1       0.743872       1.348867        0.921615      -0.173254
2       1.020434       1.172829        1.416111      -0.173254
3       0.651684       0.785546        0.674368      -0.173254
4      -1.284252      -0.869211       -1.748661      -0.173254
..           …              …              …              …
646    -0.546752      -0.481927       -1.798111      -0.173254
647     0.190747      -1.010041        0.921615      -0.173254
648    -0.546752       0.785546       -0.512422      -0.173254
649    -0.638940      -0.939626       -1.402514      -0.173254
650    -2.113938      -1.432532       -0.561872      -0.173254

     mpaa_rating_NC-17  mpaa_rating_PG  mpaa_rating_PG-13  mpaa_rating_R  \
0             -0.05547       -0.470158          -0.506322       0.988544
1             -0.05547       -0.470158           1.971992      -1.010034
2             -0.05547       -0.470158          -0.506322       0.988544
3             -0.05547        2.123679          -0.506322      -1.010034
4             -0.05547       -0.470158          -0.506322       0.988544
..                 …              …                  …              …
646           -0.05547        2.123679          -0.506322      -1.010034
647           -0.05547       -0.470158          -0.506322       0.988544
648           -0.05547       -0.470158          -0.506322       0.988544
649           -0.05547        2.123679          -0.506322      -1.010034
650           -0.05547       -0.470158           1.971992      -1.010034

     mpaa_rating_Unrated  audience_rating_Spilled  audience_rating_Upright
0              -0.288213                -0.854552                0.854552
1              -0.288213                -0.854552                0.854552
2              -0.288213                -0.854552                0.854552
3              -0.288213                -0.854552                0.854552
4              -0.288213                 1.168406               -1.168406
..                  …                        …                       …
646            -0.288213                 1.168406               -1.168406
647            -0.288213                -0.854552                0.854552
648            -0.288213                 1.168406               -1.168406
649            -0.288213                 1.168406               -1.168406
650            -0.288213                 1.168406               -1.168406
```

```
[651 rows x 11 columns]
```

#### 1.1.3 Investigate the optimal number of clusters for K-means

```
[ ]: print("All genres: ", db2['genre'].unique())
     print("Number of different movie genres in the dataset: ",␣
      ↪len(imdb_dataset['genre'].unique()))
```

```
All genres:  ['Drama' 'Comedy' 'Horror' 'Documentary' 'Action & Adventure'
 'Art House & International' 'Musical & Performing Arts'
 'Mystery & Suspense' 'Animation' 'Science Fiction & Fantasy' 'Other']
Number of different movie genres in the dataset:  11
```

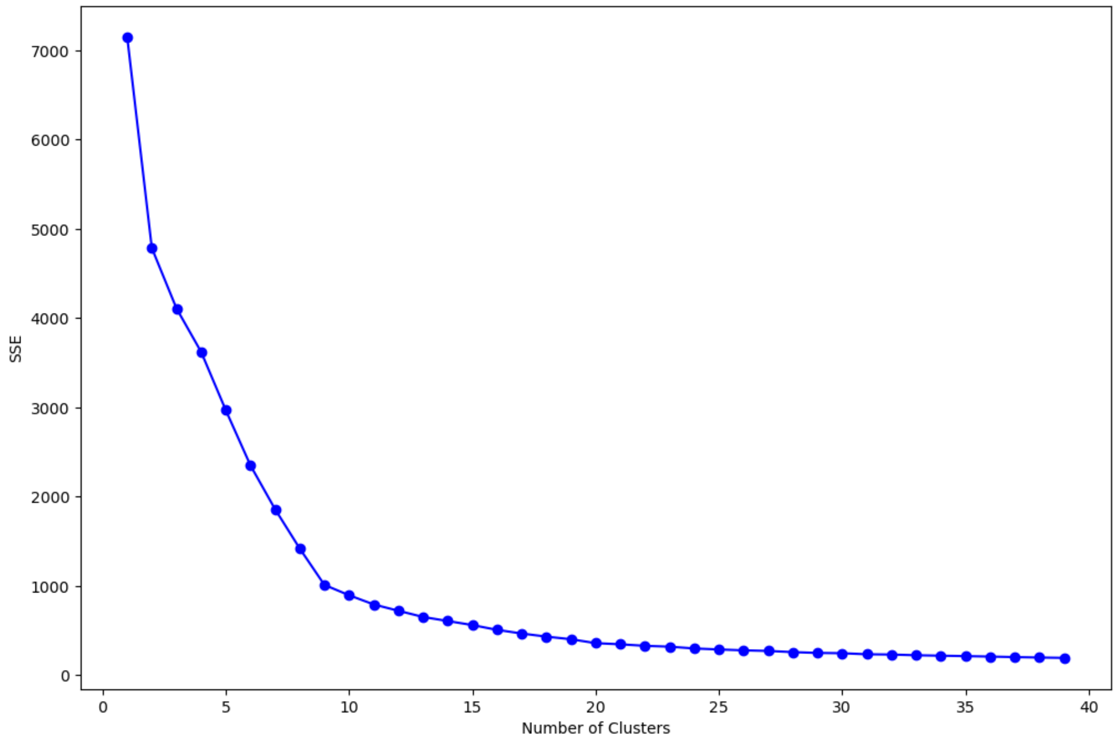
#### 1.1.4 Plot SSE vs # of clusters

With the necessary data preprocessed, we can now take it in order to apply k-means analysis with a number of clusters range from 1 to 40 for getting the amount of clusters compared to the Sum Squared Errors letting us see the distance of each data point from our range of clusters.

With that data now, we apply it with the matplotlib in order to graph the elbow method of the SSE vs # of clusters.

```
[ ]: numClusters = range(1, 40)
     SSE = []
     for k in numClusters:
         k_means = cluster.KMeans(n_clusters=k, n_init=10)
         k_means.fit(db2_preprocessed)
         SSE.append(k_means.inertia_)

     plt.xlabel('Number of Clusters')
     plt.ylabel('SSE')
     plt.plot(numClusters, SSE, marker='o', color='b')
```

```
[ ]: [<matplotlib.lines.Line2D at 0x14c4f8050>]
```




Choosing n_clusters=9 looks like a good fit on the elbow line.

[ ]:
```
n_clusters=9
```

### 1.1.5 KMeans Cluster

Split data and keep a small portion (10%) for analyzing predictions.

[ ]:
```
split_ind = int(len(db2_preprocessed) * 0.9)
data_train = db2_preprocessed[:split_ind]
data_test = db2_preprocessed[split_ind:]
print(f"Train samples: {len(data_train)}, Analysis samples: {len(data_test)}")
```

```
Train samples: 585, Analysis samples: 66
```

### 1.1.6 KMeans clustering algorithm

Using the k-means clustering model, we fit the training data above in order to find unique clusters from the dataset. Then, we can cluster each movie title to the closest cluster.

[ ]:
```
k_means = cluster.KMeans(n_clusters=n_clusters, max_iter=100, n_init=10,␣
 ↪random_state=1)
k_means.fit(data_train)
labels = k_means.labels_
print("Unique cluster ids: ", np.unique(labels))
```

```
clusters_train_df = pd.DataFrame(labels, index=db2.title[:split_ind],␣
↪columns=['Cluster ID'])
clusters_train_df
```

```
Unique cluster ids:  [0 1 2 3 4 5 6 7 8]
```

[ ]:

```
                      Cluster ID
title
Filly Brown                    1
The Dish                       4
Waiting for Guffman            1
The Age of Innocence           5
Malevolence                    0
…                            …
Hairspray                      5
Sweet Liberty                  7
Urban Cowboy                   5
Shooter                        1
Crumb                          1

[585 rows x 1 columns]
```

Append 'genre' column to analyze our clusters on input data.

[ ]:

```
# Append 'genre' column to analyze our clusters
clusters_train_df['genre'] = db2.genre[:split_ind].values
clusters_train_df.head(10)
```

[ ]:

```
                        Cluster ID        genre
title
Filly Brown                      1        Drama
The Dish                         4        Drama
Waiting for Guffman              1       Comedy
The Age of Innocence             5        Drama
Malevolence                      0       Horror
Old Partner                      2  Documentary
Lady Jane                        4        Drama
Mad Dog Time                     0        Drama
Beauty Is Embarrassing           2  Documentary
The Snowtown Murders             2        Drama
```

Analyze clusters for genre composition. Ideally clusters should show grouping of similar genres. Our clusters have good genre composition as seen below.

[ ]:

```
print("Genre composition for cluster 1")
print(clusters_train_df.groupby(['Cluster ID', 'genre']).size()[1])

print("Genre composition for cluster 2")
print(clusters_train_df.groupby(['Cluster ID', 'genre']).size()[2])
```

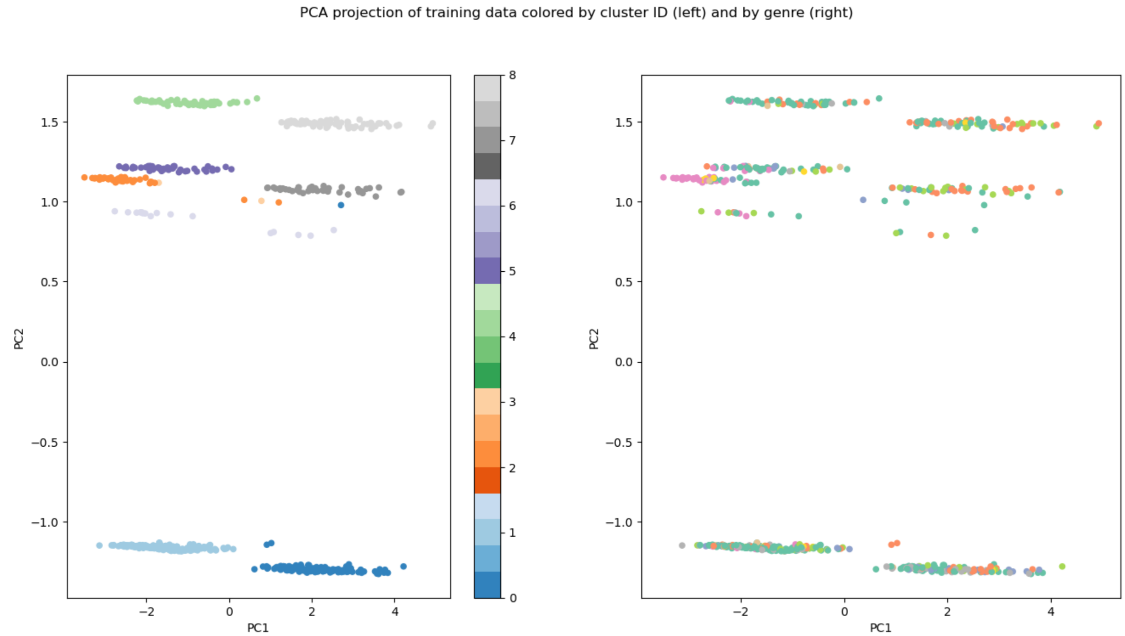
```
Genre composition for cluster 1
genre
Action & Adventure           13
Art House & International     5
Comedy                       13
Documentary                   6
Drama                       108
Horror                        2
Musical & Performing Arts     4
Mystery & Suspense           16
Other                         4
Science Fiction & Fantasy     1
dtype: int64
Genre composition for cluster 2
genre
Art House & International    3
Documentary                 27
Drama                        9
Horror                       1
Musical & Performing Arts    2
Other                        1
dtype: int64
```

### 1.1.7 Try to visualize our clusters in 2 dimensions

We project the training data to 2D with PCA and then color each sample (movie) with the cluster id color.

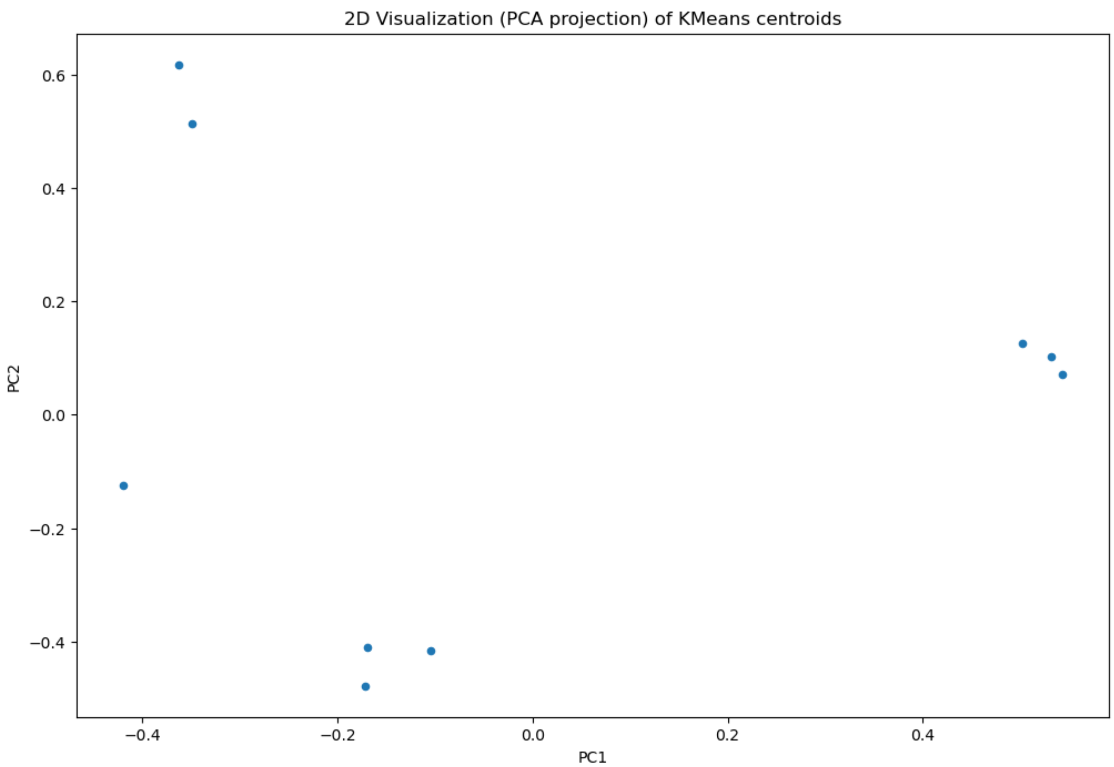
[ ]:
```
from sklearn.decomposition import PCA, KernelPCA
import seaborn as sns

plt.rcParams['figure.figsize'] = [16, 8]
fig, axes = plt.subplots(nrows=1,ncols=2)

data_train_2D = pd.DataFrame(KernelPCA(n_components=2, kernel='linear').
 ↪fit_transform(data_train), columns=['PC1', 'PC2'])
data_train_2D.plot.scatter(x='PC1', y='PC2', c=clusters_train_df['Cluster ID'],␣
 ↪colormap='tab20c', ax = axes[0], subplots=True)

color_labels = clusters_train_df['genre'].unique()
rgb_values = sns.color_palette("Set2", 11)
color_map = dict(zip(color_labels, rgb_values))
data_train_2D.plot.scatter(x='PC1', y='PC2', c=clusters_train_df['genre'].
 ↪map(color_map), \
                           title='PCA projection of training data colored by␣
 ↪cluster ID (left) and by genre (right)', \
                           ax = axes[1], subplots=True)
```

[ ]: array([<Axes: xlabel='PC1', ylabel='PC2'>], dtype=object)




```
[ ]: plt.rcParams['figure.figsize'] = [12, 8]
centroids = k_means.cluster_centers_
centroids_df = pd.DataFrame(centroids,columns=data_train.columns)
pd.DataFrame(KernelPCA(n_components=2, kernel='rbf').
 ↪fit_transform(centroids_df), columns=['PC1', 'PC2']) \
    .plot.scatter(x='PC1', y='PC2', title="2D Visualization (PCA projection) of␣
 ↪KMeans centroids")
```

[ ]: <Axes: title={'center': '2D Visualization (PCA projection) of KMeans centroids'}, xlabel='PC1', ylabel='PC2'>




### 1.1.8 Apply trained KMeans algorithm to the held out data.

[ ]:
```python
# Compute cluster labels for unseen movies using trained KMeans
labels = k_means.predict(data_test)
labels = labels.reshape(-1,1)
# Print SSE on test
print("Model inertia: ", k_means.inertia_)

# Create a dataframe that has new movies and their cluster assignment
newmovies = db[split_ind:].copy()
newmovies['Cluster ID'] = labels
print("Cluster allocation for new, unused in training movies")
newmovies
```

```
Model inertia:  889.1098459291782
Cluster allocation for new, unused in training movies
```

[ ]:
```
                              title              genre mpaa_rating  \
585                    Just Friends              Drama       PG-13
586                   Over the Edge              Drama          PG
587            Operation Dumbo Drop             Comedy          PG
588  The Postman Always Rings Twice  Mystery & Suspense          PG
589                         Monster              Drama           R
```

```
..                             …                 …       …
646              Death Defying Acts              Drama         PG
647                      Half Baked             Comedy          R
648                Dance of the Dead  Action & Adventure        R
649     Around the World in 80 Days  Action & Adventure        PG
650                             LOL             Comedy      PG-13

     imdb_rating  critics_score audience_rating  audience_score  Cluster ID
585          6.2             42         Upright              72           4
586          7.6             89         Upright              86           5
587          4.9             31         Spilled              29           7
588          6.6             83         Spilled              59           7
589          7.3             82         Upright              81           1
..           …            …             …             …         …
646          5.9             44         Spilled              26           7
647          6.7             29         Upright              81           1
648          5.9             80         Spilled              52           0
649          5.8             31         Spilled              34           7
650          4.2             17         Spilled              51           8

[66 rows x 8 columns]
```

## 1.2 Hierarchical Analysis on the IMDB dataset

[ ]:
```
imdb_dataset
```

[ ]:
```
      id                            title    title_type               genre  \
0      1                      Filly Brown  Feature Film               Drama
1      2                         The Dish  Feature Film               Drama
2      3               Waiting for Guffman  Feature Film              Comedy
3      4             The Age of Innocence  Feature Film               Drama
4      5                      Malevolence  Feature Film              Horror
..   …                            …           …                 …
646  647               Death Defying Acts  Feature Film               Drama
647  648                       Half Baked  Feature Film              Comedy
648  649                Dance of the Dead  Feature Film  Action & Adventure
649  650   Around the World in 80 Days  Feature Film  Action & Adventure
650  651                              LOL  Feature Film              Comedy

     runtime mpaa_rating                      studio  thtr_rel_year  \
0       80.0           R        Indomina Media Inc.           2013
1      101.0       PG-13      Warner Bros. Pictures           2001
2       84.0           R     Sony Pictures Classics           1996
3      139.0          PG          Columbia Pictures           1993
4       90.0           R   Anchor Bay Entertainment           2004
..       …         …                       …            …
646     97.0          PG         Genius Productions           2008
```

```
647     82.0           R        Universal Pictures          1998
648     87.0           R  Grindhouse Entertainment          2008
649    120.0          PG      Buena Vista Pictures          2004
650     97.0       PG-13          Lionsgate Films           2012

     thtr_rel_month  thtr_rel_day  …  best_dir_win  top200_box  \
0                 4            19  …            no          no
1                 3            14  …            no          no
2                 8            21  …            no          no
3                10             1  …           yes          no
4                 9            10  …            no          no
..              …             …  …           …          …
646               7            11  …            no          no
647               1            16  …            no          no
648               3             9  …            no          no
649               6            16  …            no         yes
650               5             4  …            no          no

              director             actor1               actor2  \
0     Michael D. Olmos     Gina Rodriguez         Jenni Rivera
1           Rob Sitch          Sam Neill      Kevin Harrington
2    Christopher Guest  Christopher Guest      Catherine O'Hara
3      Martin Scorsese   Daniel Day-Lewis     Michelle Pfeiffer
4          Stevan Mena      Samantha Dark    R. Brandon Johnson
..                 …                 …                    …
646  Gillian Armstrong         Guy Pearce  Catherine Zeta-Jones
647        Tamra Davis     Dave Chappelle         Guillermo Diaz
648       Gregg Bishop      Jared Kusnitz      Greyson Chadwick
649       Frank Coraci        Jackie Chan          Steve Coogan
650       Liza Azuelos        Miley Cyrus            Demi Moore

                  actor3              actor4               actor5  \
0    Lou Diamond Phillips       Emilio Rivera  Joseph Julian Soria
1      Patrick Warburton            Tom Long      Genevieve Mooy
2           Parker Posey         Eugene Levy          Bob Balaban
3           Winona Ryder    Richard E. Grant          Alec McCowen
4        Brandon Johnson       Heather Magee       Richard Glover
..                    …                   …                    …
646        Timothy Spall       Saoirse Ronan          Jack Bailey
647           Jim Breuer    Harland Williams          Rachel True
648       Chandler Darby  Carissa Capobianco       Randy McDowell
649         Ewen Bremner         Robert Fyfe          Ian McNeice
650        Ashley Greene       Douglas Booth       Adam G. Sevani

                               imdb_url  \
0    http://www.imdb.com/title/tt1869425/
1    http://www.imdb.com/title/tt0205873/
```

```
2    http://www.imdb.com/title/tt0118111/
3    http://www.imdb.com/title/tt0106226/
4    http://www.imdb.com/title/tt0388230/
..                                   …
646  http://www.imdb.com/title/tt0472071/
647  http://www.imdb.com/title/tt0120693/
648  http://www.imdb.com/title/tt0926063/
649  http://www.imdb.com/title/tt0327437/
650  http://www.imdb.com/title/tt1592873/

                                                  rt_url
0         //www.rottentomatoes.com/m/filly_brown_2012/
1                     //www.rottentomatoes.com/m/dish/
2      //www.rottentomatoes.com/m/waiting_for_guffman/
3         //www.rottentomatoes.com/m/age_of_innocence/
4     //www.rottentomatoes.com/m/10004684-malevolence/
..                                                 …
646     //www.rottentomatoes.com/m/death_defying_acts/
647             //www.rottentomatoes.com/m/half_baked/
648  //www.rottentomatoes.com/m/1203339-dance_of_th…
649  //www.rottentomatoes.com/m/around_the_world_in…
650               //www.rottentomatoes.com/m/lol_2011/

[651 rows x 33 columns]
```

### 1.3 Single Link

Here we apply encoding to the dataset in order to change the text string values into a integer value.

[ ]:
```
encode_text_index(imdb_dataset, 'title_type')
encode_text_index(imdb_dataset, 'mpaa_rating')
encode_text_index(imdb_dataset, 'critics_rating')
encode_text_index(imdb_dataset, 'audience_rating')
encode_text_index(imdb_dataset, 'best_pic_nom')
encode_text_index(imdb_dataset, 'best_pic_win')
encode_text_index(imdb_dataset, 'best_actor_win')
encode_text_index(imdb_dataset, 'best_actress_win')
encode_text_index(imdb_dataset, 'best_dir_win')
encode_text_index(imdb_dataset, 'top200_box')
imdb_dataset
```

[ ]:
```
      id                   title  title_type               genre  \
0      1             Filly Brown           1               Drama
1      2                The Dish           1               Drama
2      3     Waiting for Guffman           1              Comedy
3      4  The Age of Innocence           1               Drama
4      5             Malevolence           1              Horror
```

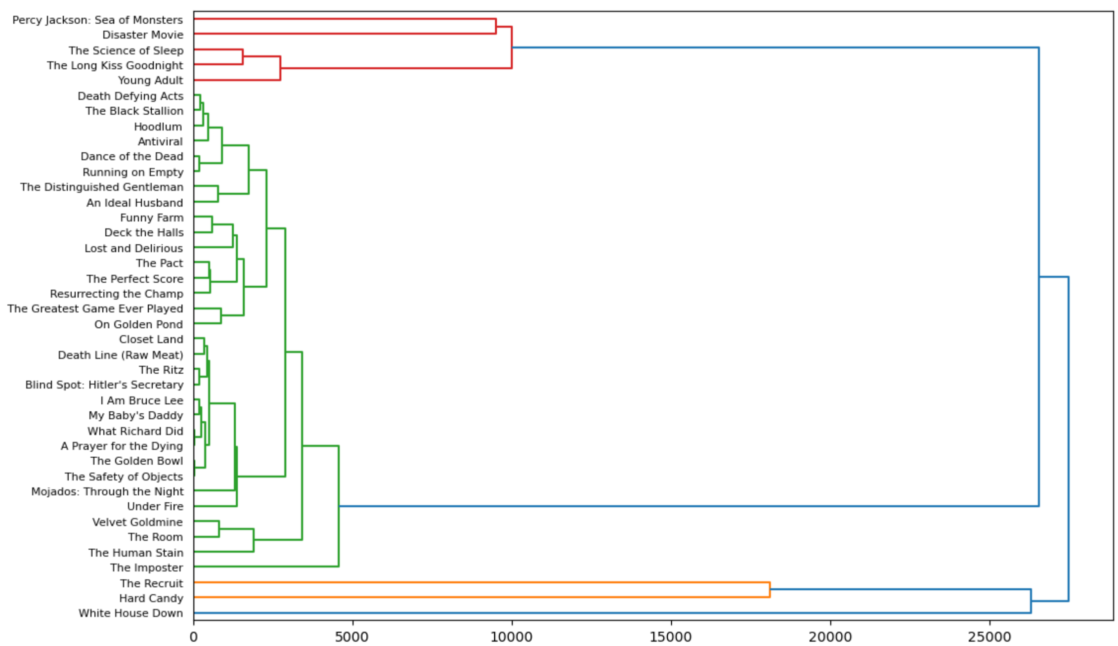
```
..   …                            …          …                  …
646  647          Death Defying Acts          1              Drama
647  648                  Half Baked          1             Comedy
648  649           Dance of the Dead          1  Action & Adventure
649  650  Around the World in 80 Days          1  Action & Adventure
650  651                         LOL          1             Comedy

     runtime  mpaa_rating                     studio  thtr_rel_year  \
0       80.0            4       Indomina Media Inc.           2013
1      101.0            3     Warner Bros. Pictures           2001
2       84.0            4    Sony Pictures Classics           1996
3      139.0            2         Columbia Pictures           1993
4       90.0            4  Anchor Bay Entertainment           2004
..       …          …                        …              …
646     97.0            2       Genius Productions           2008
647     82.0            4       Universal Pictures           1998
648     87.0            4  Grindhouse Entertainment           2008
649    120.0            2      Buena Vista Pictures           2004
650     97.0            3           Lionsgate Films           2012

     thtr_rel_month  thtr_rel_day  …  best_dir_win  top200_box  \
0                 4            19  …             0           0
1                 3            14  …             0           0
2                 8            21  …             0           0
3                10             1  …             1           0
4                 9            10  …             0           0
..              …           …  …           …         …
646               7            11  …             0           0
647               1            16  …             0           0
648               3             9  …             0           0
649               6            16  …             0           1
650               5             4  …             0           0

              director              actor1                actor2  \
0     Michael D. Olmos     Gina Rodriguez          Jenni Rivera
1            Rob Sitch          Sam Neill      Kevin Harrington
2    Christopher Guest  Christopher Guest      Catherine O'Hara
3      Martin Scorsese   Daniel Day-Lewis     Michelle Pfeiffer
4          Stevan Mena      Samantha Dark    R. Brandon Johnson
..                 …                …                     …
646  Gillian Armstrong         Guy Pearce  Catherine Zeta-Jones
647        Tamra Davis     Dave Chappelle        Guillermo Diaz
648       Gregg Bishop     Jared Kusnitz      Greyson Chadwick
649       Frank Coraci        Jackie Chan          Steve Coogan
650       Liza Azuelos        Miley Cyrus            Demi Moore

                     actor3              actor4              actor5  \
```

```
0    Lou Diamond Phillips       Emilio Rivera  Joseph Julian Soria
1       Patrick Warburton            Tom Long       Genevieve Mooy
2            Parker Posey         Eugene Levy          Bob Balaban
3            Winona Ryder    Richard E. Grant         Alec McCowen
4         Brandon Johnson       Heather Magee       Richard Glover
..                    …                   …                    …
646         Timothy Spall       Saoirse Ronan          Jack Bailey
647            Jim Breuer    Harland Williams          Rachel True
648         Chandler Darby  Carissa Capobianco       Randy McDowell
649          Ewen Bremner         Robert Fyfe          Ian McNeice
650          Ashley Greene       Douglas Booth       Adam G. Sevani

                                    imdb_url  \
0    http://www.imdb.com/title/tt1869425/
1    http://www.imdb.com/title/tt0205873/
2    http://www.imdb.com/title/tt0118111/
3    http://www.imdb.com/title/tt0106226/
4    http://www.imdb.com/title/tt0388230/
..                                     …
646  http://www.imdb.com/title/tt0472071/
647  http://www.imdb.com/title/tt0120693/
648  http://www.imdb.com/title/tt0926063/
649  http://www.imdb.com/title/tt0327437/
650  http://www.imdb.com/title/tt1592873/

                                                 rt_url
0         //www.rottentomatoes.com/m/filly_brown_2012/
1                     //www.rottentomatoes.com/m/dish/
2      //www.rottentomatoes.com/m/waiting_for_guffman/
3         //www.rottentomatoes.com/m/age_of_innocence/
4     //www.rottentomatoes.com/m/10004684-malevolence/
..                                                 …
646     //www.rottentomatoes.com/m/death_defying_acts/
647             //www.rottentomatoes.com/m/half_baked/
648  //www.rottentomatoes.com/m/1203339-dance_of_th…
649  //www.rottentomatoes.com/m/around_the_world_in…
650               //www.rottentomatoes.com/m/lol_2011/

[651 rows x 33 columns]
```

Limiting the dataset so clustering plot is more readable & displaying the dendrogram for single link hierarchical clustering.

[ ]:
```
from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
%matplotlib inline
```

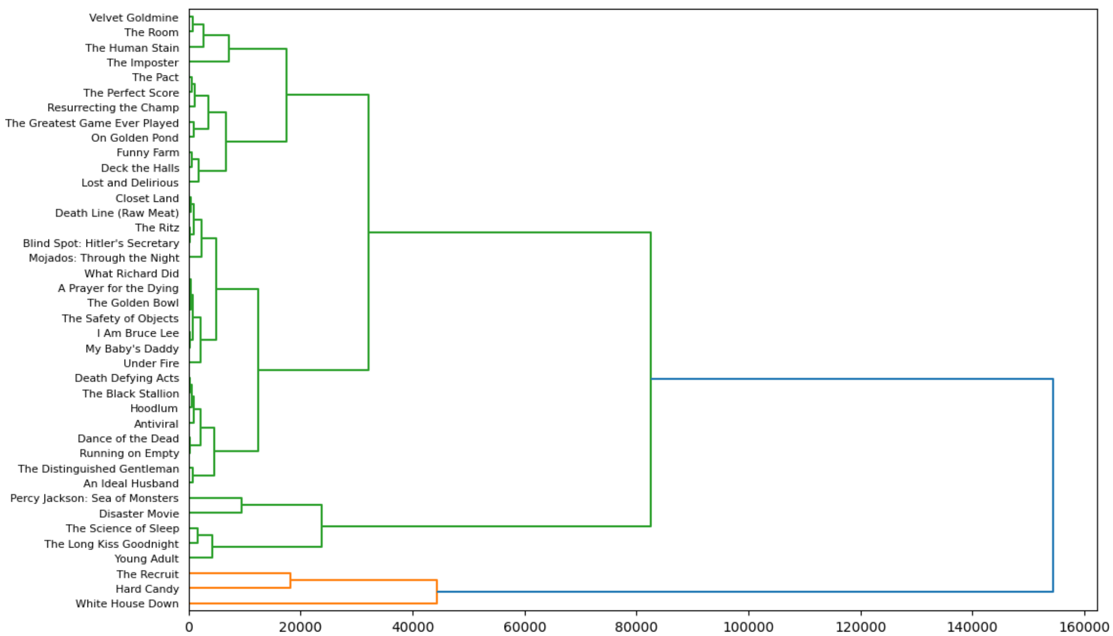
```
Y = imdb_dataset['genre']
X = imdb_dataset.drop(['id', 'title','genre', 'runtime', 'studio',␣
↪'thtr_rel_year', 'thtr_rel_month', 'thtr_rel_day', 'dvd_rel_year',␣
↪'dvd_rel_month', 'dvd_rel_day',
               'director', 'actor1', 'actor2', 'actor3', 'actor4', 'actor5',␣
↪'imdb_url', 'rt_url'],axis=1)

# Minimizing the rows by choosing 40 random movies
names = imdb_dataset['title'].sample(n=40, random_state=0)
X = X.sample(n=40, random_state=0)

Z = hierarchy.linkage(X.values, 'single')
dn = hierarchy.dendrogram(Z,labels=names.tolist(),orientation='right')
```




### 1.3.1 Complete Link

This time, we do hierarchical clustering with complete linkage which calculates gets the max distance between clusters then displaying the dendrogram.

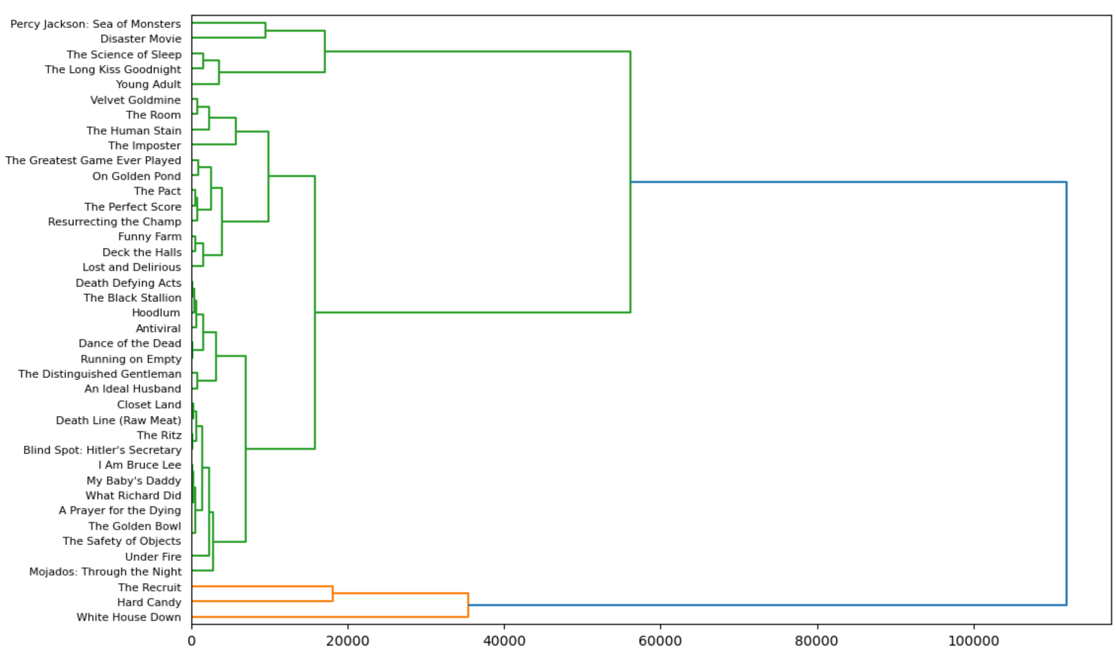
[ ]:
```
Z = hierarchy.linkage(X.values, 'complete')
dn = hierarchy.dendrogram(Z,labels=names.tolist(),orientation='right')
```




### 1.3.2 Group Average

Lastly, we do the group average method of hierarchical clustering getting average distance instead between cluster points.

[ ]:
```
Z = hierarchy.linkage(X.values, 'average')
dn = hierarchy.dendrogram(Z,labels=names.tolist(),orientation='right')
```



# 2 Part 2: Text Mining

### 2.0.1 Dataset for text mining:

[ ]:
```
#From the assignment 4 directions
text_dataset = [ 'Now for manners use has company believe parlors.',
'Least nor party who wrote while did. Excuse formed as is agreed admire so on␣
↪result parish.',
'Put use set uncommonly announcing and travelling. Allowance sweetness␣
↪direction to as necessary.',
'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 property␣
↪exercise of. ' ,
'Dissimilar comparison no terminated devonshire no literature on. Say most yet␣
↪head room such just easy. ']
```

### 2.0.2 Count Vector Implementation

Vectorizer picks out unique words and places their count in a vector. We then take vectorizer and format it to a matrix using a transform function. When we print out the matrix, it will display each unique word in a column and how many times each document (row) has used them.

[ ]:
```
import sklearn.feature_extraction.text as sk_text

#min_df is set to 2 to keep the matrix from being too cluttered.
vectorizer = sk_text.CountVectorizer(min_df=2)
#vectorizer = sk_text.CountVectorizer(stop_words = 'english')

#min_df: ignore terms that have a document frequency < min_df.

#format the vectorizer into a readable matrix.
matrix = vectorizer.fit_transform(text_dataset)

print(type(matrix))          # Compressed Sparse Row matrix
print(matrix.toarray())        #  convert it to numpy array

print(vectorizer.get_feature_names_out())
```

```
<class 'scipy.sparse._csr.csr_matrix'>
[[0 0 0 0 0 0 0 0 1 0]
 [1 1 1 0 1 0 1 0 0 0]
```

```
 [0 1 0 0 0 0 0 0 1 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 1 1 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 0 1 1]
 [1 0 0 0 1 0 0 0 0 1]
 [0 0 0 0 0 2 0 0 0 0]
 [0 0 0 0 0 2 1 0 0 0]]
['admire' 'as' 'did' 'excellent' 'formed' 'no' 'on' 'therefore' 'use'
 'warmth']
```

### 2.0.3 Tfidf Vector Implementation

TFIDF calculates how relevant a word is to a text. Vectorizer takes the unique words and evaluates them based on the number of times a word appears compared to the frequency in the dataset. We then format the vector into a matrix and print out the result.

[ ]:
```
vectorizer = sk_text.TfidfVectorizer(
                             #stop_words='english',
                             #max_features = 1000,
                             min_df=2)
#min_df is set to 2 to prevent the matrix from being too cluttered.

#max_features:  build a vocabulary that only consider the top max_features␣
↪features ordered by term frequency across the corpus.

matrix = vectorizer.fit_transform(text_dataset)

print(type(matrix))          # Compressed Sparse Row matrix
print(matrix.toarray())        #  convert it to numpy array
np.set_printoptions(precision=4)
print(vectorizer.get_feature_names_out())
```

```
<class 'scipy.sparse._csr.csr_matrix'>
[[0.         0.         0.         0.         0.         0.
  0.         0.         1.         0.        ]
 [0.4472136  0.4472136  0.4472136  0.         0.4472136  0.
  0.4472136  0.         0.         0.        ]
 [0.         0.75262077 0.         0.         0.         0.
  0.         0.         0.65845424 0.        ]
 [0.         0.         0.         1.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.57735027 0.57735027 0.         0.
  0.         0.57735027 0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         1.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.65845424 0.75262077]
```

```
 [0.57735027 0.         0.         0.         0.57735027 0.
  0.         0.         0.         0.57735027]
 [0.         0.         0.         0.         0.         1.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.89442719
  0.4472136  0.         0.         0.        ]]
['admire' 'as' 'did' 'excellent' 'formed' 'no' 'on' 'therefore' 'use'
 'warmth']
```

2.4) Tfidf (term frequency-inverse document frequency) is a measure of how frequent a word appears in a set of documents. It is generally used in text analysis algorithms and for document searching. For example, Google search uses Tfidf for text preprocessing.

# 3 Part 3: Artificial Neural Network (ANN)

## 3.1 ANN Implementation

In this section, we will be performing ANN techniques on the Admission dataset.

### 3.1.1 Useful functions

[ ]:
```
import pandas as pd
def change_to_binary_values(df, col_name):
    df[col_name] = (df[col_name] > df[col_name].median()).astype('int')

#Function to normalize columns
def normalize_numeric_minmax(df, name):
        df[name] = ((df[name] - df[name].min()) / (df[name].max() - df[name].
 ↪min())).astype(np.float32)

# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for␣
 ↪red,green,blue)
# def encode_text_dummy(df, name):
#     dummies = pd.get_dummies(df[name])
#     for x in dummies.columns:
#         dummy_name = "{}-{}".format(name, x)
#         df[dummy_name] = dummies[x]
#     df.drop(name, axis=1, inplace=True)

# Encode text values to indexes(i.e. [1],[2],[3] for red,green,blue).
def encode_text_index(df, name):
    le = preprocessing.LabelEncoder()
    df[name] = le.fit_transform(df[name])
    return le.classes_

# Convert a Pandas dataframe to the x,y inputs that TensorFlow needs
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) else target_type
    # 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.float32)
```

Importing the Admission dataset and displaying it

[ ]:
```
import pandas as pd
admission_dataset = pd.read_csv("./Admission_Predict_Ver1.
↪1_small_data_set_for_Linear_Regression-1.csv")
admission_dataset = admission_dataset.drop(columns=['Serial No.'])
admission_dataset
```

[ ]:
```
     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]
```

#### 3.1.2 Preprocessing foe ANN: Normalize numerical predictors and binarize the targets for classification.

For a numerical variable X that takes values in the range [a, b] where a < b, we normalize the measurements by subtracting a and dividing by b − a.

[ ]:
```
normalize_numeric_minmax(admission_dataset, 'GRE Score')
normalize_numeric_minmax(admission_dataset, 'TOEFL Score')
normalize_numeric_minmax(admission_dataset, 'University Rating')
normalize_numeric_minmax(admission_dataset, 'SOP')
normalize_numeric_minmax(admission_dataset, 'LOR ')
normalize_numeric_minmax(admission_dataset, 'CGPA')
change_to_binary_values(admission_dataset, 'Chance of Admit ')
admission_dataset
```

[ ]:
```
     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]
```

Now all input features should be in [0, 1] range.

Down below, we will be splitting up the admission dataset into training and testing that will be used to calculate our Mean Sum of Error

```
[ ]: X = admission_dataset.drop('Chance of Admit ', axis=1)
     y = admission_dataset['Chance of Admit ']
```

Our testing size is sitting at 20% of the dataset

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
      ↪random_state=42)
```

```
[ ]: scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

#### 3.1.3 Classification with sklearn MLPClassifier with 2 hidden layers

Once we have stadardized our training and test dataset, we then apply MLP (Multi-layer perceptron) Classification which comes from the neural network sklearn library.

```
[ ]: mlp = MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000, random_state=42)
     mlp.fit(X_train_scaled, y_train)
```

```
[ ]: MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000, random_state=42)
```

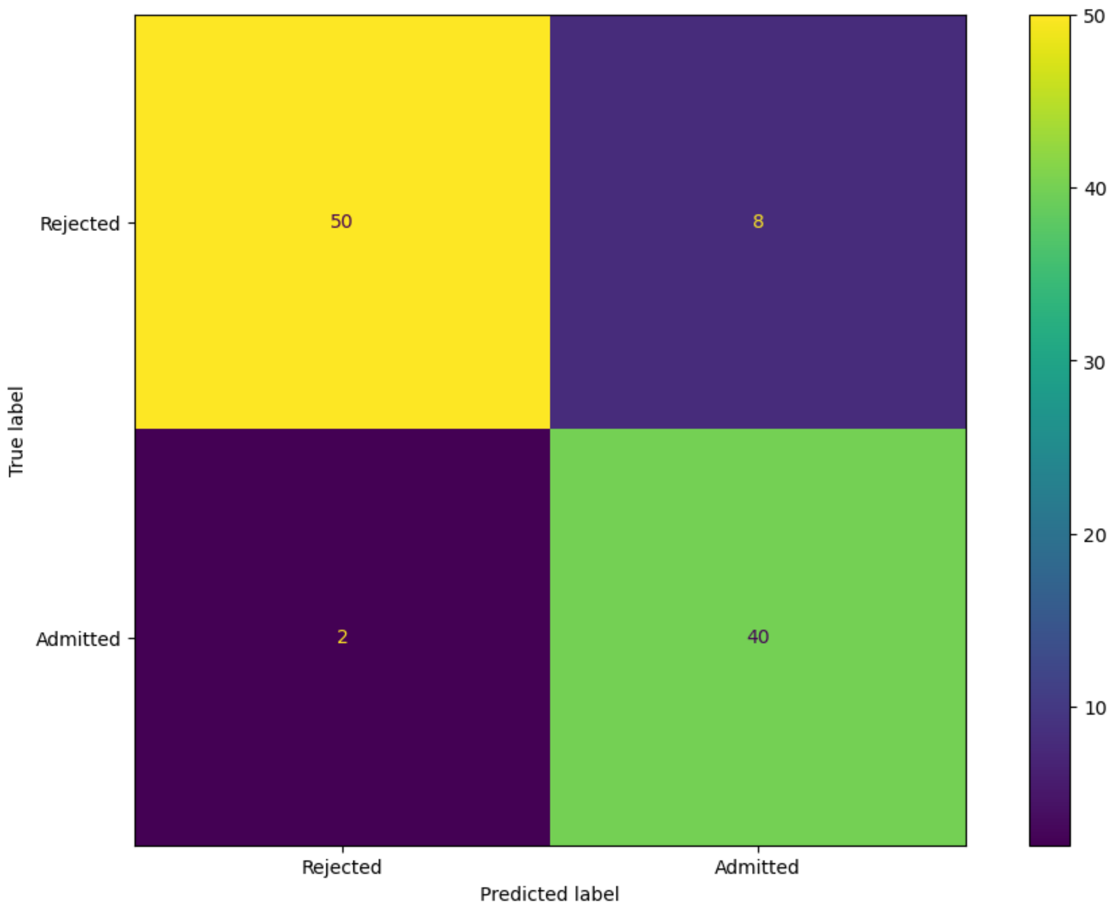
```
[ ]: y_pred_score = mlp.predict(X_test_scaled)
     y_pred = y_pred_score > 0.5
```

**MLP Classification report**

```
[ ]: print('Accuracy on test data is %.2f' % (accuracy_score(y_test, y_pred)))
     print(classification_report(y_test, y_pred))
     cm = confusion_matrix(y_test, y_pred)
     disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Rejected',␣
      ↪'Admitted'])
     disp.plot()
     plt.show()
```

```
Accuracy on test data is 0.90
              precision    recall  f1-score   support

           0       0.96      0.86      0.91        58
           1       0.83      0.95      0.89        42

    accuracy                           0.90       100
```

```
   macro avg       0.90      0.91      0.90       100
weighted avg       0.91      0.90      0.90       100
```




Using the prediction that uses the MLP Classifier, we then find the Mean sum squarred error. The mean sum squared error shows to us that the error for this dataset when comparing both the tested variable and the predicted variable has a low error value when predicting.

[ ]:
```
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.1
```

### 3.1.4 Tensorflow Keras ANN Prediction and Classification

Down below, we will be importing keras for using ANN methods. Make sure that keras and tensorflow is installed in order for the code to work.

[ ]:
```
import tensorflow as tf
```

```
admission_dataset = pd.read_csv("./Admission_Predict_Ver1.
↪1_small_data_set_for_Linear_Regression-1.csv")
admission_dataset = admission_dataset.drop(columns="Serial No.")
admission_dataset
```

[ ]:
```
     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]
```

Converting Chance of Admit to become binary values. This is due to ANN using [0, 1]

[ ]:
```
change_to_binary_values(admission_dataset, 'Chance of Admit ')
admission_dataset
```

[ ]:
```
     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                   1
1                   1
2                   0
3                   1
4                   0
..                ...
495                 1
496                 1
497                 1
498                 1
499                 1

[500 rows x 8 columns]
```

Converting the Chance of Admit column into 'yes' and 'no' values and storing it into a variable called classes

[ ]:
```
admission_dataset['Chance of Admit '].replace((1, 0), ('yes', 'no'),␣
 ↪inplace=False)
classes = encode_text_index(admission_dataset, 'Chance of Admit ')
classes
```

[ ]:
```
array([0, 1])
```

### 3.1.5 Normalize numerical columns and separate features and targets for training.

[ ]:
```
normalize_numeric_minmax(admission_dataset, 'GRE Score')
normalize_numeric_minmax(admission_dataset, 'TOEFL Score')
normalize_numeric_minmax(admission_dataset, 'University Rating')
normalize_numeric_minmax(admission_dataset, 'SOP')
normalize_numeric_minmax(admission_dataset, 'LOR ')
normalize_numeric_minmax(admission_dataset, 'CGPA')
```

Create a test data set that will take 40 random rows from the admission set

[ ]:
```
# Choosing a random sample of 40 rows for our testing
split_index = 460
test_data = admission_dataset[split_index:]
train_data = admission_dataset[:split_index]
test_data.head(10)
```

[ ]:
```
     GRE Score  TOEFL Score  University Rating    SOP    LOR       CGPA  \
460       0.58     0.464286               0.75  0.750  0.875  0.596154
461       0.22     0.357143               0.50  0.375  0.250  0.426282
```

```
462        0.34     0.464286                 0.75  0.500  0.500  0.365385
463        0.28     0.535714                 0.50  0.625  0.500  0.339744
464        0.16     0.178571                 0.25  0.250  0.500  0.131410
465        0.30     0.142857                 0.75  0.500  0.875  0.467949
466        0.48     0.250000                 0.75  0.625  0.875  0.618590
467        0.56     0.321429                 1.00  0.625  1.000  0.634615
468        0.66     0.642857                 0.75  0.750  1.000  0.666667
469        0.72     0.785714                 0.75  0.750  0.625  0.756410

     Research  Chance of Admit
460         1                1
461         1                0
462         0                0
463         0                0
464         0                0
465         0                0
466         1                0
467         1                1
468         1                1
469         1                1
```

[ ]:
```python
X, y = to_xy(train_data, 'Chance of Admit ')
testX, testY = to_xy(test_data, 'Chance of Admit ')
```

[ ]:
```python
print(X.shape)
print(y.shape)
```

```
(460, 7)
(460, 2)
```

Create a Neural network with 2 hidden Dense layers with ReLU activations and a final Softmax layer to predict one hot encoded targets.

[ ]:
```python
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8, input_dim = X.shape[1], activation='relu'))
model.add(tf.keras.layers.Dropout(rate=0.5))
model.add(tf.keras.layers.Dense(4, activation='relu'))
model.add(tf.keras.layers.Dense(2, activation='softmax'))
```

Define the loss and optimizer and fit model to training data

[ ]:
```python
model.compile(loss='categorical_crossentropy',optimizer='adam')
model.fit(X, y, verbose=0, epochs=1000, batch_size=1000)
```

[ ]:
```
<keras.src.callbacks.History at 0x17e58a690>
```

Using the model for generating our predicted values

```
[ ]: pred = model.predict(testX)
     print(pred[0])
```

```
2/2 [==============================] - 0s 2ms/step
[0.3983 0.6017]
2/2 [==============================] - 0s 2ms/step
[0.3983 0.6017]
```

```
[ ]: pred = np.argmax(pred, axis=1)
     true = np.argmax(testY, axis=1)
```

Outputting our class (which is the Chance of Admit column) and observing our predicted set from our actual set

```
[ ]: print("Predicted classes: ", classes[pred])
     print("True classes: ", classes[true])
```

```
Predicted classes:  [1 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0
0 0 0 0 0 1 1
 1 0 1]
True classes:  [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]
```

Generating the accuracy from our ANN technique as well as the classification report. As the value is somewhat above average, this tells us that our predicted values are in line with our actual dataset.

```
[ ]: print('Accuracy on test data is %.2f' % (accuracy_score(true, pred)))
     print(classification_report(true,pred))
```

```
Accuracy on test data is 0.93
              precision    recall  f1-score   support

           0       0.91      0.95      0.93        22
           1       0.94      0.89      0.91        18

    accuracy                           0.93        40
   macro avg       0.93      0.92      0.92        40
weighted avg       0.93      0.93      0.92        40
```