Data analysis Project 3

Applying machine learning methods to movie ratings data

NYU CDS, Fall 2021

Introduction to Data Science: Project 3

student netid: aks9136

Dataset description

This dataset features ratings data of 400 movies from 1097 research participants.

- 1st row: Headers (Movie titles/questions) – note that the indexing in this list is from 1
- Row 2-1098: Responses from individual participants
- Columns 1-400: These columns contain the ratings for the 400 movies (0 to 4, and missing)
- Columns 401-421: These columns contain self-assessments on sensation seeking behaviors (1-5)
- Columns 422-464: These columns contain responses to personality questions (1-5)
- Columns 465-474: These columns contain self-reported movie experience ratings (1-5)
- Column 475: Gender identity (1 = female, 2 = male, 3 = self-described)
- Column 476: Only child (1 = yes, 0 = no, -1 = no response)
- Column 477: Movies are best enjoyed alone (1 = yes, 0 = no, -1 = no response)

Note that we did most of the data munging for you already (e.g. Python interprets commas in a csv file as separators, so we removed all commas from movie titles), but you still need to handle missing data.

Setup

In [ ]:

```
########################
###   READ IN DATA   ###
### IMPORT LIBRARIES ###
########################

import pandas as pd
import numpy as np
from tqdm import tqdm
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge
from sklearn import linear_model
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm
```

```
movies_raw = pd.read_csv('movieReplicationSet.csv')
movies_raw.head(5)
```

Out[ ]:

	The Life of David Gale (2003)	Wing Commander (1999)	Django Unchained (2012)	Alien (1979)	Indiana Jones and the Last Crusade (1989)	Snatch (2000)	Rambo: First Blood Part II	Fargo (1996)	Let the Right One In (2008)	Black Swan (2010)
0	NaN	NaN	4.0	NaN	3.0	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	1.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	2.0	NaN	3.0	NaN	NaN	NaN	NaN	4.0
4	NaN	NaN	3.5	NaN	0.5	NaN	0.5	1.0	NaN	0.0

5 rows × 477 columns

In [ ]:

```
########################
### DATA DEFINITION ###
########################

# Define x-features
x_vars = movies_raw.iloc[: , 420:474]
print('Full x_vars :', len(x_vars))
x_vars = x_vars.dropna()
print('Drop NA x_vars :', len(x_vars))
x_vars.head(5)

# Break up features

# Personality features
x_personality = x_vars.iloc[: , :44]
# Movie watching features
x_movie_experience = x_vars.iloc[: , 44:]
```

```
Full x_vars : 1097
Drop NA x_vars : 968
```

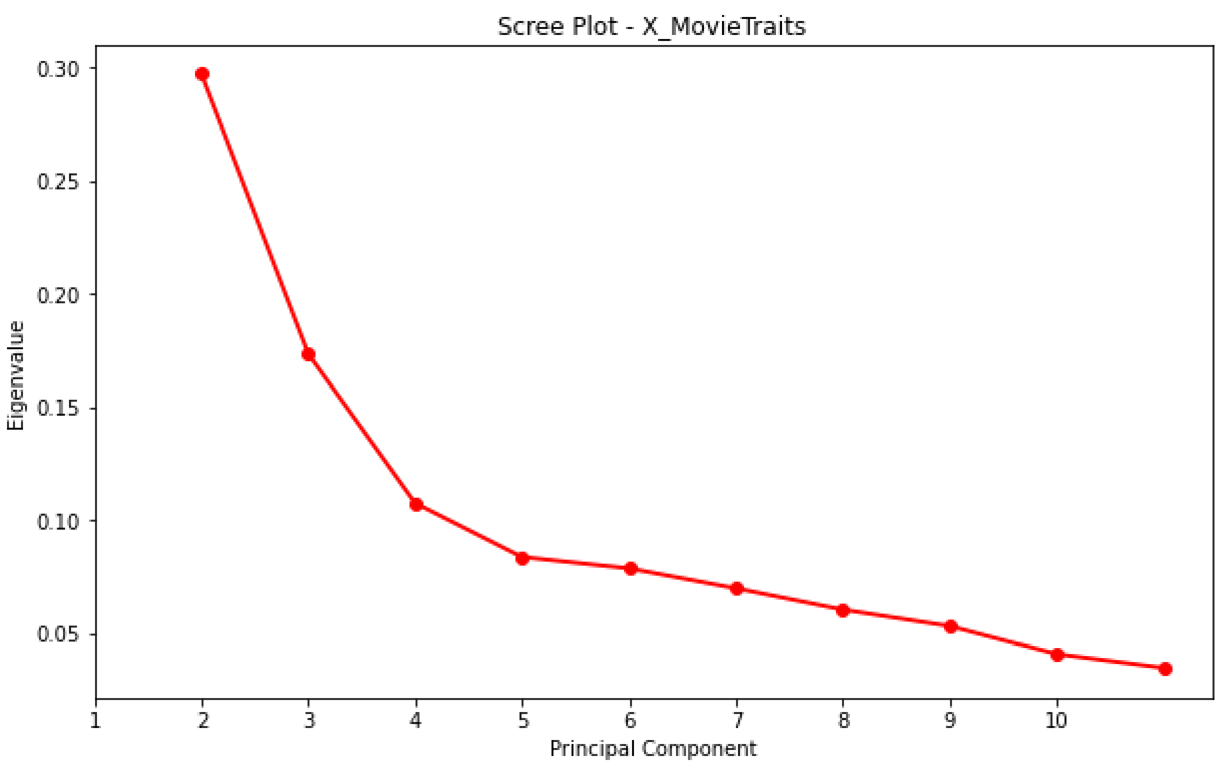
Problem 1a and 1b

a) Determine the number of factors (principal components) that you will interpret meaningfully (by a criterion of your choice – but make sure to name that criterion). Include a Scree plot in your answer.

b) Semantically interpret what those factors represent (hint: Inspect the loadings matrix). Explicitly name the factors you found and decided to interpret meaningfully in 1a). Be creative

In [ ]:

```
###########################################
### Choosing # of Principle Components ###
###########################################

# Generate Scree Plots - Personality Traits
df_norm = (x_personality - x_personality.mean()) / (x_personality.max() - x_pers

A,B,C=np.linalg.svd(df_norm)
eigen_values=B**2/np.sum(B**2)
figure=plt.figure(figsize=(10,6))
sing_vals=np.arange(len(eigen_values)) + 1
plt.plot(sing_vals,eigen_values, 'ro-', linewidth=2)
plt.title('Scree Plot - X_Personality')
xi = list(range(len(sing_vals)))
plt.xticks(xi, sing_vals)
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')
plt.show()

# Generate Scree Plots - Movie Watching Traits
df_norm = (x_movie_experience - x_movie_experience.mean()) / (x_movie_experience

A,B,C=np.linalg.svd(df_norm)
eigen_values=B**2/np.sum(B**2)
figure=plt.figure(figsize=(10,6))
sing_vals=np.arange(len(eigen_values)) + 1
plt.plot(sing_vals,eigen_values, 'ro-', linewidth=2)
plt.title('Scree Plot - X_MovieTraits')
xi = list(range(len(sing_vals)))
plt.xticks(xi, sing_vals)
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')
plt.show()
```



In [ ]:

```
##############################
### Investigate Components ###
##############################

### Personality ###
pca = PCA(n_components=3)
principalComponents_personality = pd.DataFrame(pca.fit_transform(x_personality))

loadings = pd.DataFrame(pca.components_.T, index=x_personality.columns)
#print(loadings[2].sort_values(ascending=False).head(10))
```

```
#print('---------------------')
#rint(loadings[2].sort_values(ascending=True).head(10))


# 0 - Socially Unenergetic
# 1 - Reserved
# 2 - Self-Assured

# 8 components - lots of overlap from component to component
# 6 components - 4th component was bizarre, plus overlap
# Chosing 3 components

### Movie Preferences ###
pca = PCA(n_components=3)
principalComponents_movie = pd.DataFrame(pca.fit_transform(x_movie_experience))

loadings = pd.DataFrame(pca.components_.T, index=x_movie_experience.columns)
#print(loadings[2].sort_values(ascending=False).head(10))
#print('---------------------')
#rint(loadings[2].sort_values(ascending=True).head(10))

# 0 - Emotional Immersion
# 1 - Low Plot Retention
# 2 - Physical/Auditory Reactivity

# 5 components -- got too vague
```

In [ ]:

```
x_reduced = pd.concat([principalComponents_personality, principalComponents_movi
x_reduced.columns  = ['Socially_Unenergetic', 'Reserved', 'Self_Assured', 'Emoti
x_reduced.head(5)
```

Out[ ]:

	Socially_Unenergetic	Reserved	Self_Assured	Emotional_Immersion	Low_Plot_Retention	Phys
0	2.075634	3.192395	-0.464452	-0.279485	-1.694314	
1	-0.037979	-0.273356	0.093231	2.533484	-0.590147	
2	-1.080811	-1.886126	-3.323927	-1.952771	-2.184912	
3	1.979029	-1.927534	1.159626	1.448319	0.287622	
4	6.561127	1.657595	-0.462845	-0.480588	-2.491528	

ANSWER:

a) Ultimately, based on the scree plot and the interperability of the results, we choose 6 features to reduce down to. The criterion was simply a number of dimensions that was reasonable close to the dropoff point on the scree plot AND yielded clear interpretations of each component. In other words, we went with the number of dimensions that was most statistically correct while maintaining interpretability.

b) We semantically interpret our components using the following titles: 'Socially_Unenergetic', 'Reserved', 'Self_Assured', 'Emotional_Immersion', 'Low_Plot_Retention', 'Physical_Auditory_Reactivity'

Problem 2

Plot the data from columns 421-474 in the new coordinate system, where each dot represents a person, and the axes represent the factors you found in 1). Hint: If you identified more than 2 meaningful factors, it is a good idea to create several 2D (X vs. Y) subplots for better interpretability.

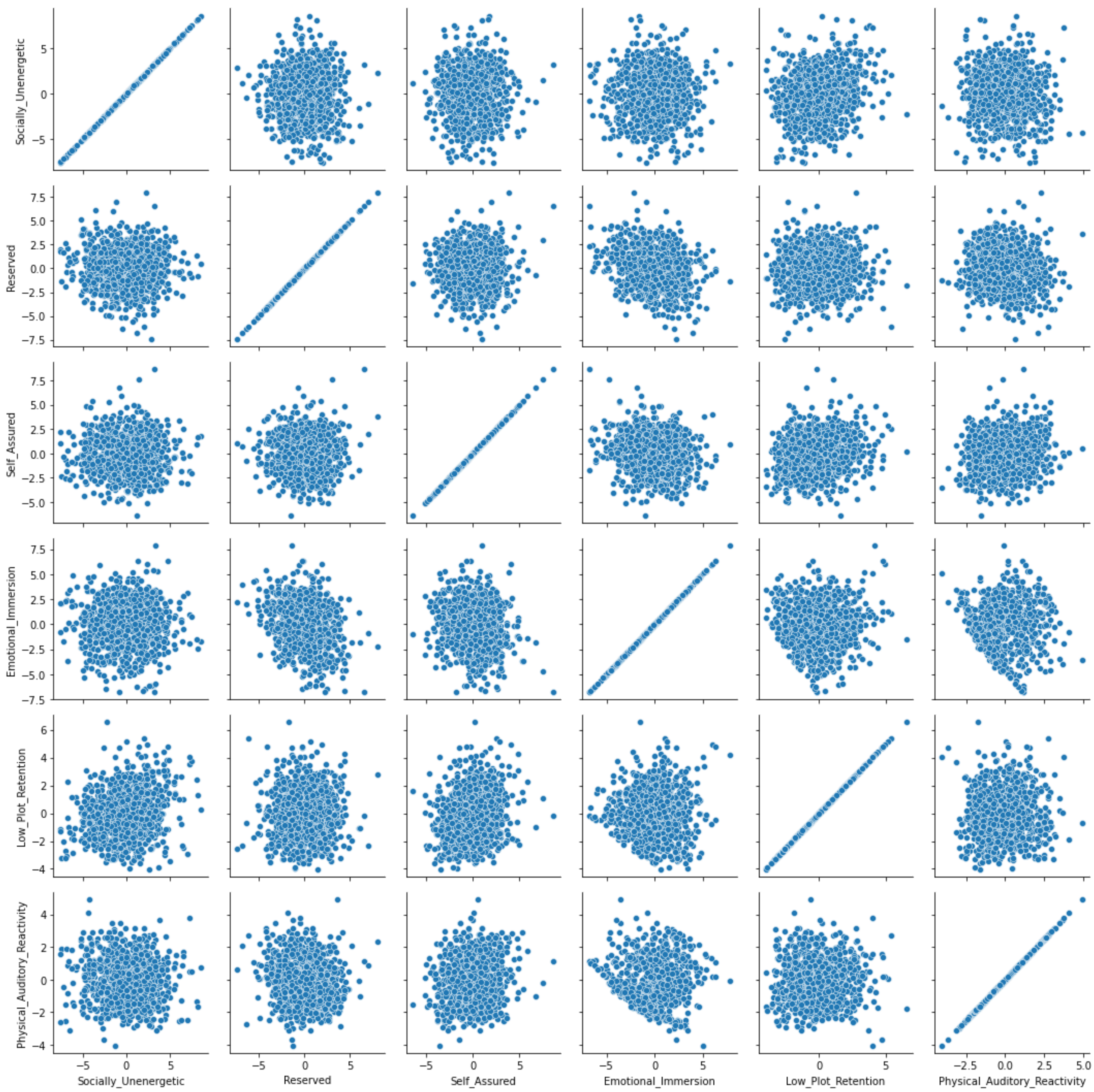
In [ ]:

```
#######################
### GENERATE PLOTS ###
#######################

print('PAIRWISE SCATTER PLOTS')
g = sns.PairGrid(x_reduced)
g.map(sns.scatterplot)
```

```
PAIRWISE SCATTER PLOTS
```

Out[ ]:

```
<seaborn.axisgrid.PairGrid at 0x7fde306a44f0>
```



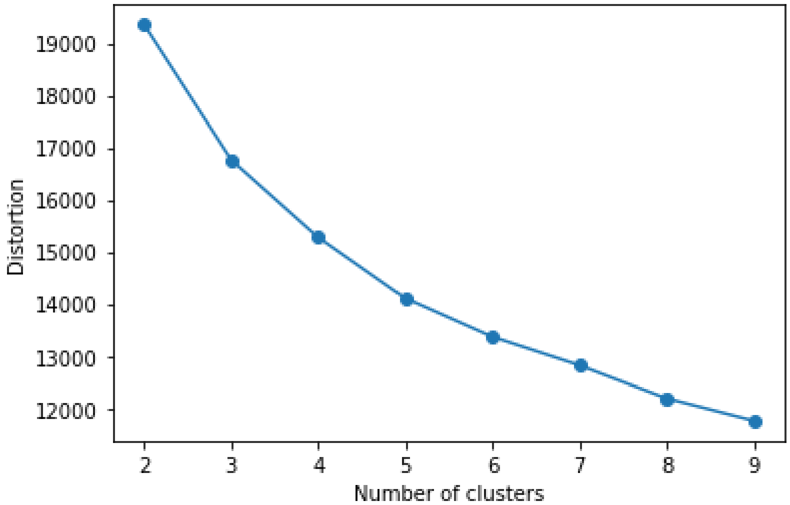
Problem 3

Identify clusters in this new space. Use a method of your choice (e.g. kMeans, DBScan, hierarchical clustering) to do so. Determine the optimal number of clusters and identify which cluster a given user is part of

In [ ]:

```
##################
### CLUSTERING ###
##################

# K Means
loss = []
for i in range(2, 10):
    cluster = KMeans(
        n_clusters=i, init='random',
        n_init=10, max_iter=300,
        tol=1e-04, random_state=0
    )
    cluster.fit(x_reduced)
    cluster_labels = cluster.fit_predict(x_reduced)
    loss.append(cluster.inertia_)

    centers = cluster.cluster_centers_

    score = silhouette_score(x_reduced, cluster_labels)
    print("For n_clusters = {}, silhouette score is {})".format(i, score))

# Plot
plt.plot(range(2, 10), loss, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.show()

# Chosing 2 Clusters
cluster = KMeans(n_clusters=2, init='random',
        n_init=10, max_iter=300,
        tol=1e-04, random_state=0)
cluster_labels = cluster.fit_predict(x_reduced)
```

```
For n_clusters = 2, silhouette score is 0.17584348784777004)
For n_clusters = 3, silhouette score is 0.15749054370557888)
For n_clusters = 4, silhouette score is 0.14774619837217662)
For n_clusters = 5, silhouette score is 0.1450790794267551)
For n_clusters = 6, silhouette score is 0.13766030630239648)
For n_clusters = 7, silhouette score is 0.1302228789622809)
For n_clusters = 8, silhouette score is 0.13334844725844716)
For n_clusters = 9, silhouette score is 0.13126670142065536)
```



In [ ]:

```
x_reduced_with_label = x_reduced
x_reduced_with_label['cluster'] = cluster_labels
x_reduced_with_label.head(5)
```

Out[ ]:

	Socially_Unenergetic	Reserved	Self_Assured	Emotional_Immersion	Low_Plot_Retention	Phys
0	2.075634	3.192395	-0.464452	-0.279485	-1.694314	
1	-0.037979	-0.273356	0.093231	2.533484	-0.590147	
2	-1.080811	-1.886126	-3.323927	-1.952771	-2.184912	
3	1.979029	-1.927534	1.159626	1.448319	0.287622	
4	6.561127	1.657595	-0.462845	-0.480588	-2.491528	