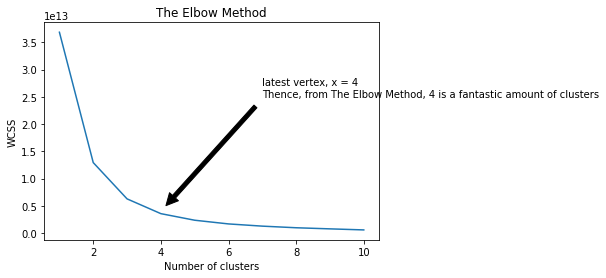
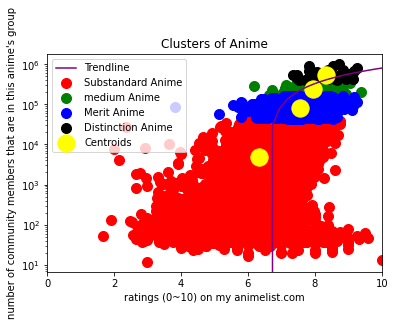
# -\*- coding: utf-8 -\*-  
*"""assignment 3.ipynb  
  
Automatically generated by Colaboratory.  
  
Original file is located at  
 https://colab.research.google.com/drive/1cBzcS027YXnz8\_g2DhLWsE9nse2hIRNS  
  
Assignment III  
  
Due: June 5, 2022.   
Pick a dataset of your choice. From UCI classification datasets   
(https://archive.ics.uci.edu/ml/datasets.php?format=&task=cla&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table)  
Use the RandomForest method for classification.   
Take from UCI dataset from the clustering section and run k-means algorithm on it.   
Submit:  
 1) code in Python   
2) a word file summarizing the result include some measure of performance such as confusion matrix, and accuracy.  
  
Enjoy!!  
"""*# Commented out IPython magic to ensure Python compatibility.  
#constructing dataframe  
from google.colab import drive  
drive.mount('/content/drive')  
import numpy as np  
import matplotlib.pyplot as plt   
import pandas as pd   
# %matplotlib inline  
#classification  
#clustering  
df = pd.read\_csv("/content/drive/MyDrive/Colab Notebooks/anime.csv", sep=",")  
df['episodes'] = pd.to\_numeric(df["episodes"], errors='coerce').fillna(1000).astype(int)  
df.episodes[df.episodes <= 25] = 0  
df.episodes[df.episodes > 25] = 1  
df = df.dropna(how="any")  
print("\n")  
print(f"The whole dataframe looks like:\n {df}.")  
print("\n")  
print("The dataframe information is: ")  
print(f"{df.info()}.")  
print("\n")  
print(f"The dataframe shape is: {df.shape}.")  
X\_all = df.iloc[:, 5:7]  
y = df["episodes"]  
  
#K-Means  
from sklearn.cluster import KMeans  
wcss = []  
for i in range(1,11):  
 k\_means = KMeans(n\_clusters= i, init = "k-means++", random\_state = 42)  
 k\_means.fit(X\_all)  
 wcss.append(k\_means.inertia\_)  
plt.plot(range(1,11), wcss)  
plt.title("The Elbow Method")  
plt.xlabel("Number of clusters")  
plt.ylabel("WCSS")  
plt.yscale("linear")  
plt.annotate('latest vertex, x = 4\nThence, from The Elbow Method, 4 is a fantastic amount of clusters', xy=(4, 0.4\*(10\*\*13)), xytext=(7, 2.5\*10\*\*13), arrowprops=dict(facecolor='black', shrink=0.05))  
plt.show()  
print(f"Thence, from The Elbow Method, 4 is a fantastic amount of clusters.")  
k\_means = KMeans(n\_clusters = 4, init = "k-means++", random\_state = 42)  
X\_all = np.array(X\_all).reshape(-1,2)  
y\_k\_means = k\_means.fit\_predict(X\_all)  
plt.scatter(X\_all[y\_k\_means == 0, 0], X\_all[y\_k\_means ==0, 1], s= 100, c= "red", label = "Substandard Anime")  
plt.scatter(X\_all[y\_k\_means == 1, 0], X\_all[y\_k\_means ==1, 1], s= 100, c= "green", label = "medium Anime")  
plt.scatter(X\_all[y\_k\_means == 2, 0], X\_all[y\_k\_means ==2, 1], s= 100, c= "blue", label = "Merit Anime")  
plt.scatter(X\_all[y\_k\_means == 3, 0], X\_all[y\_k\_means ==3, 1], s= 100, c= "black", label = "Distinction Anime")  
plt.scatter(k\_means.cluster\_centers\_[:,0], k\_means.cluster\_centers\_[:,1], s=300, c = "yellow", label = "Centroids")  
plt.title("Clusters of Anime")  
plt.xlabel("ratings (0~10) on my animelist.com")  
plt.ylabel("number of community members that are in this anime's group")  
plt.xscale("linear")  
plt.yscale("log")  
plt.xlim(0,10)  
plt.ylim()  
from sklearn.linear\_model import LinearRegression  
lin\_reg = LinearRegression()  
array\_x = np.array([k\_means.cluster\_centers\_[0,0], k\_means.cluster\_centers\_[1,0], k\_means.cluster\_centers\_[2,0], k\_means.cluster\_centers\_[3,0]])  
array\_y = np.array([k\_means.cluster\_centers\_[0,1], k\_means.cluster\_centers\_[1,1], k\_means.cluster\_centers\_[2,1], k\_means.cluster\_centers\_[3,1]])  
array\_x = array\_x.reshape(4,1)  
array\_y = array\_y.reshape(4,1)  
lin\_reg.fit(array\_x, array\_y)  
intercept = lin\_reg.intercept\_  
gradient = lin\_reg.coef\_  
x0 = np.linspace(0,10)  
y0 = x0\*gradient+intercept  
y0 = y0.reshape(50,)  
plt.plot(x0,y0,label="Trendline", c='purple')  
plt.legend()  
plt.show()  
#evaluation  
from matplotlib import cm  
from sklearn.metrics import silhouette\_samples  
cluster\_labels = np.unique(y\_k\_means)  
n\_clusters = cluster\_labels.shape[0]  
silhouette\_vals = silhouette\_samples(X\_all, y\_k\_means, metric='euclidean')  
y\_ax\_lower, y\_ax\_upper = 0, 0  
yticks = []  
for i, c in enumerate(cluster\_labels):  
 c\_silhouette\_vals = silhouette\_vals[y\_k\_means == c]  
 c\_silhouette\_vals.sort()  
 y\_ax\_upper += len(c\_silhouette\_vals)  
 color = cm.jet(i / n\_clusters)  
 plt.barh(range(y\_ax\_lower, y\_ax\_upper), c\_silhouette\_vals, height=1.0, edgecolor='none', color=color)  
 yticks.append((y\_ax\_lower + y\_ax\_upper) / 2)  
 y\_ax\_lower += len(c\_silhouette\_vals)  
silhouette\_avg = np.mean(silhouette\_vals)  
plt.axvline(silhouette\_avg, color="red", linestyle="--")  
plt.yticks(yticks, cluster\_labels + 1)  
plt.ylabel('Cluster')  
plt.xlabel('Silhouette coefficient')  
plt.show()  
  
#Random Forest  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_all, y, test\_size = 0.5, random\_state = 0)  
from sklearn.preprocessing import StandardScaler   
sc = StandardScaler()  
X\_train = np.array(X\_train).reshape(-1,2)  
X\_train = sc.fit\_transform(X\_train)  
X\_test = np.array(X\_test).reshape(-1,2)  
X\_test = sc.transform(X\_test)  
from sklearn.ensemble import RandomForestClassifier  
rfc = RandomForestClassifier(n\_estimators = 500, criterion="entropy", random\_state = 0)  
rfc.fit(X\_train, y\_train)  
y\_pred\_rfc = rfc.predict(X\_test)  
#train set  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_train, y\_train  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:,0].min()-1, stop = X\_set[:,0].max()+1, step = 0.01), np.arange(start=X\_set[:,1].min() -1, stop = X\_set[:,1].max() +1, step= 0.01))  
plt.contourf(X1, X2, rfc.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha=0.75, cmap= ListedColormap(("red","blue")))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i,j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(("red", "blue"))(i), label = j)  
plt.title("Random Forest Classification (Trainning Set)")  
plt.xlabel("ratings (0~10) on my animelist.com")  
plt.ylabel("number of community members that are in this anime's group")  
plt.legend()  
plt.show()  
#test set  
from matplotlib.colors import ListedColormap  
X\_set, y\_set = X\_test, y\_test  
X1, X2 = np.meshgrid(np.arange(start = X\_set[:,0].min()-1, stop = X\_set[:,0].max()+1, step = 0.01), np.arange(start=X\_set[:,1].min() -1, stop = X\_set[:,1].max() +1, step= 0.01))  
plt.contourf(X1, X2, rfc.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha=0.75, cmap= ListedColormap(("red","blue")))  
plt.xlim(X1.min(), X1.max())  
plt.ylim(X2.min(), X2.max())  
for i,j in enumerate(np.unique(y\_set)):  
 plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(("red", "blue"))(i), label = j)  
plt.title("Random Forest Classification (Test Set)")  
plt.xlabel("ratings (0~10) on my animelist.com")  
plt.ylabel("number of community members that are in this anime's group")  
plt.legend()  
plt.show()  
#evaluation  
from sklearn import metrics  
print(metrics.classification\_report(y\_pred\_rfc , y\_test))  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred\_rfc)  
import seaborn as sns   
sns.heatmap(cm.T, square=True, annot=True, fmt='d', cbar=False)  
plt.xlabel('true label')  
plt.ylabel('predicted label');  
plt.show()

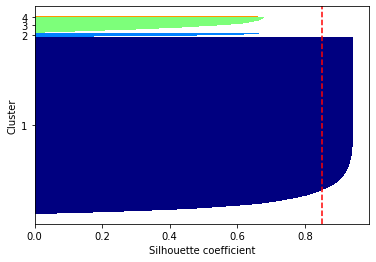
Results:

**For k-means:**



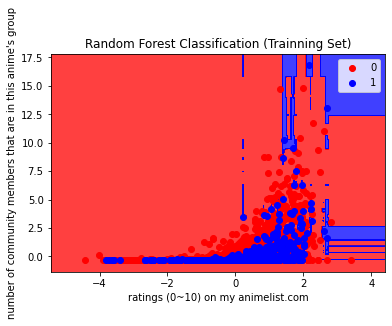
Thence, from The Elbow Method, 4 is a fantastic amount of clusters.

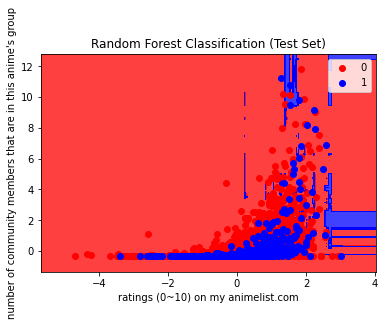




Using the Elbow method, I could automatically what number of clusters best suits my data, so by looking at the vertex, I find 4 clusters to set, and when I deploy them, there is a nice trend created. The performance, however, is not satisfying because the clustering of 4 areas are not evenly distributed, as we can see from the Silhouette model.

**For Random Forest:**





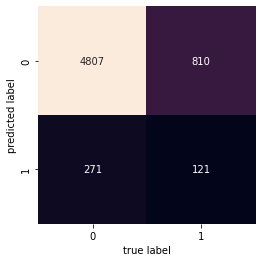
precision recall f1-score support

0 0.95 0.86 0.90 5617

1 0.13 0.31 0.18 392

accuracy 0.82 6009

macro avg 0.54 0.58 0.54 6009

weighted avg 0.89 0.82 0.85 6009

The weighted accuracy of 89% seems decently high, but it is not perfect. Although, on the first glance, the ListedColormaps look identical, but details are quite different, even the scale is different. Nonetheless, this method is the very accurate one available to people.