Final Project

Our goal in this problem is to practice Kmeans clustering using two methods of initialization: using hierarchical clustering to get k clusters from a subset of the data to start and the other is through random initialization. We will then compare the two different implementaions using adjuster_rand_score(from sklearn). I believe the true k number of clusters will be 4 and that if I compare the different clustering between random initialization and hierarchical , they should have a positive adjusted_rand_score (from sklearn).

I have played video games almost my entire life (earliest I remember was 7), my drive of competition with my brother pushed me further down that rabbit hole. In all my time gaming, not once have I ever asked which video games are most closely related to each other? To answer that question we will be working with a videogame dataset that comes from Kaggle([https://www.kaggle.com/ashaheedq/video-games-sales-2019?select=vgsales-12-4-2019.csv (https://www.kaggle.com/ashaheedq/video-games-sales-2019?select=vgsales-12-4-2019.csv)](https://www.kaggle.com/ashaheedq/video-games-sales-2019?select=vgsales-12-4-2019.csv)) originating from vgchartz.com. The dataset contains data of videogames ranks, genre, ESRB rating (Everyone, Teen, Mature, Etc.), platform, publisher, critics score, user scores, sales (NA-North America, PAL- Europe, JP-Japan, Global) and more. It contains a total of 23 columns and 37102 unique values. It seems to be a fairly large data that will definitely need some data cleaning: as I could already see some columns that will either be not useful or to sparse. Here is each column

- Rank: Ranking of overall sales
- Name: Name of the game
- BaseName: If two games come out on different platforms (consider merging?)
- Genre: Genre of the game
- ESRB Rating: ESRB Rating of the game
- Platform: (PC, PS4, XBoxOne, Wii,etc.)
- Publisher: of the game
- Developer: of the game
- Critic Score: of game (1-10)
- User Score: of game (1-10)
- Total Shipped: Total shipped copies
- Global_Sales: worldwide sales (in millions)
- NA_Sales: Sales in North America (in millions)
- PAL_Sales: Sales in Europe (in millions)
- JP_Sales: Sales in Japan (in millions)
- Other_Sales: Sales in the rest of the world (in millions)
- Year:Year of release of the game
- VGChartz_Score: empty (full of NaNs)
- Last_Update: when the numbers where last updated
- url: takes to location of game on VGchartz.com
- status: if it is currently available to play (all 1s)
- Vgchartzscore: the score give by VGchartz on the game (sparse)]
- img-url: picture of game from the VGchartz website (dont seem to be live links)

I am going to attempt to cluster this data using **7** of these columns:
'Critic_Score','User_Score','NA_Sales','PAL_Sales','JP_Sales','Other_Sales','Year'

I have decided on these columns as they are numeric and have wide range of factors for the game.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
```

In [2]:
```
df = pd.read_csv ('vgsales-12-4-2019.csv', encoding= 'UTF-8', header=0) #read
 data in
```

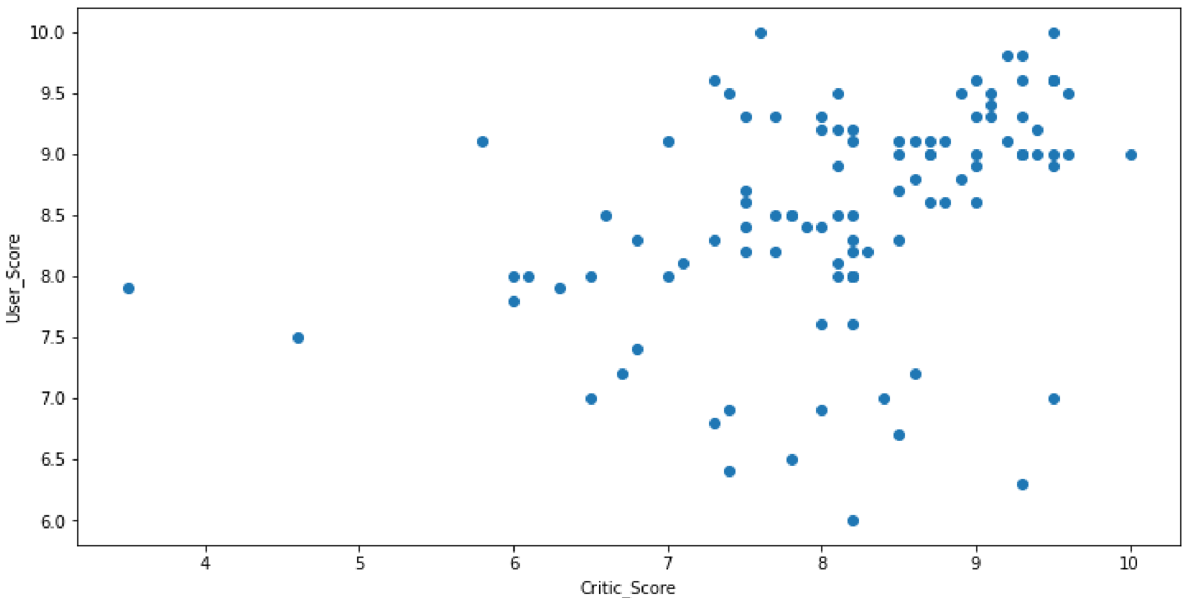
In [3]:
```
df = df[df['NA_Sales'].notna()] #drop nans in all numerical categories
df = df[df['PAL_Sales'].notna()]
df = df[df['JP_Sales'].notna()]
df = df[df['Other_Sales'].notna()]
df = df[df['Critic_Score'].notna()]
df.drop(columns=['VGChartz_Score', 'url','status','img_url','Last_Update','Tot
al_Shipped','Vgchartzscore'],inplace=True) #drop unnecessary columns
df = df[df['User_Score'].notna()] #for including user score as a column in kme
ans, just reduces amount to 103 vs 1308 when done drop Nans
df.shape
```

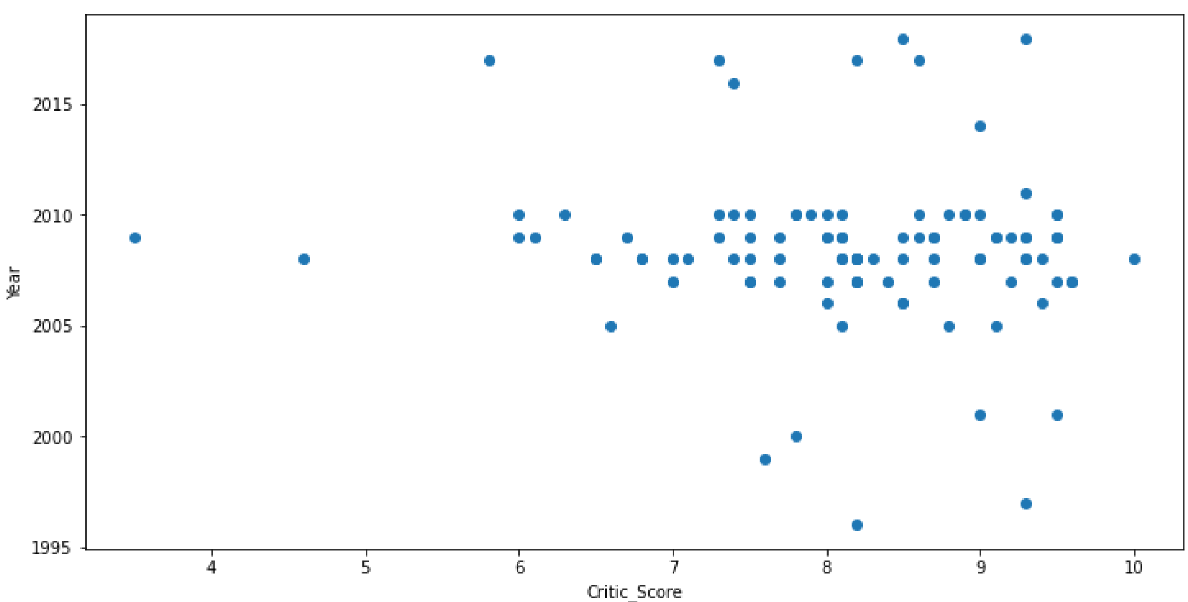
Out[3]:
```
(103, 16)
```

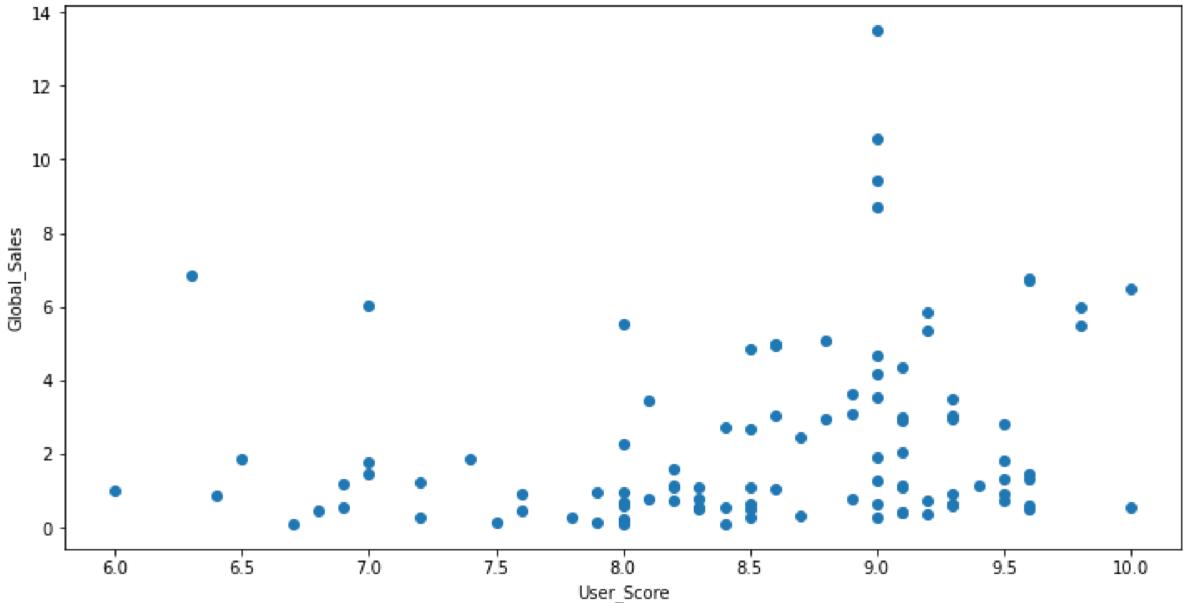
Now after cleaning the data and dropping all a NaNs from data we are left with a size of 103 videogames. Lets take a look at a few plots of different columns we will be working on.

In [4]:
```
#cars= sm.datasets.get_rdataset('mtcars').data
print(df.head(3))

#example plot of mpg x hp
def printScatter(x,y,df):
    fig, ax= plt.subplots(figsize=(12,6))
    plt.scatter(df[x], df[y])
    ax.set_xlabel(x)
    ax.set_ylabel(y)
printScatter('Critic_Score','User_Score',df)
printScatter('Critic_Score','Year',df)
printScatter('User_Score','NA_Sales',df)
```

```
    Rank                          Name                        basename  \
49    50  Call of Duty: Modern Warfare 2  call-of-duty-modern-warfare-2
79    80              Grand Theft Auto IV            grand-theft-auto-iv
94    95  Call of Duty 4: Modern Warfare  call-of-duty-4-modern-warfare

     Genre ESRB_Rating Platform       Publisher       Developer  \
49  Shooter           M     X360      Activision   Infinity Ward
79   Action           M      PS3  Rockstar Games  Rockstar North
94  Shooter           M     X360      Activision   Infinity Ward

    Critic_Score  User_Score  Global_Sales  NA_Sales  PAL_Sales  JP_Sales  \
49           9.5         9.0         13.53      8.54       3.63      0.08
79          10.0         9.0         10.57      4.79       3.73      0.44
94           9.6         9.0          9.41      5.98       2.39      0.13

    Other_Sales    Year
49         1.28  2009.0
79         1.62  2008.0
94         0.91  2007.0
```







In [5]:
```
dfData=df[['Critic_Score','User_Score','NA_Sales','PAL_Sales','JP_Sales','Othe
r_Sales','Year']].copy()
```

The first thing we will need to do is find the best k to use for our problem. Since Random initialization is a bit easier to start with in terms of coding complexity, we will use this to find our best k by graphing a range from k=2:10(maybe more) by increments of 2 displayed against within-cluster sum of square (wss).

In [6]:
```
def visualize(clusters,df,x,y):
    fig, ax = plt.subplots(figsize=(8,4))
    print(clusters)
    plt.scatter(df[x], df[y], c=clusters)
    ax.set_xlabel(x)
    ax.set_ylabel(y);
def dist(x1, x2):
    return np.sqrt(np.sum((x1-x2)**2))
def kMeansRand(K,df,n):

    clocs=np.random.choice(range(n), size=K)
    centroids = np.array([df.iloc[clocs[i]] for i in range(K)]) # initialize f
or any k value
    #print(centroids)
    dists = [0]*K
    clustersn = np.array([-1]*n)

    for i in range(50): #might change to tolerance/change check when I use for
comparison with hierarchical clustering
        #update classifications
        for j in range(n):
            dists = [dist(df.iloc[j], cent) for cent in centroids]
            assignment = dists.index(min(dists))
            clustersn[j] = assignment#update centroids

        for j in range(K):

            centroids[j]=[np.sum(df.iloc[clustersn==j,0])/len(df.iloc[clusters
n==j,0]),\
                          np.sum(df.iloc[clustersn==j,1])/len(df.iloc[clusters
n==j,1]),\
                          np.sum(df.iloc[clustersn==j,2])/len(df.iloc[clusters
n==j,2]),\
                          np.sum(df.iloc[clustersn==j,3])/len(df.iloc[clusters
n==j,3]),\
                         np.sum(df.iloc[clustersn==j,4])/len(df.iloc[clustersn
==j,4]),\
                         np.sum(df.iloc[clustersn==j,5])/len(df.iloc[clustersn
==j,5]),\
                         np.sum(df.iloc[clustersn==j,6])/len(df.iloc[clustersn
==j,6])]

    #WCCS = [0]*K#Within Cluster Sum of Squares (WCSS) average between each da
ta point in a cluster
    #counter = [0]*K
    avgDistBetweenClust_DataP=0
    #sums each euclidean distance from cluster point is assigned
    for j in range(n):
        dists = [dist(df.iloc[j], cent) for cent in centroids]
        avgDistBetweenClust_DataP+=dists[clustersn[j]]
        #print(j,dists,clustersn[j])
        #WCCS[clustersn[j]]+= dists[clustersn[j]]
        #counter[clustersn[j]]+=1
    avgDistBetweenClust_DataP/n
    #divide by amount of points in each cluster will the WCCS
    #WCCS = [WCCS[i]/counter[i] for i in range(K)]
```

```
    #WCCS_avg= np.mean(WCCS)
    #print(WCCS, counter,WCCS_avg)
    #print(clustersn)

    #WCCS_avg=0
    return K,clustersn,avgDistBetweenClust_DataP
```

In [7]:
```
dfData_norm=dfData.copy()
#need to normalize data
means=dfData_norm.mean()
sd=dfData_norm.std(ddof=1)
dfData_norm=(dfData_norm-means)/sd
```

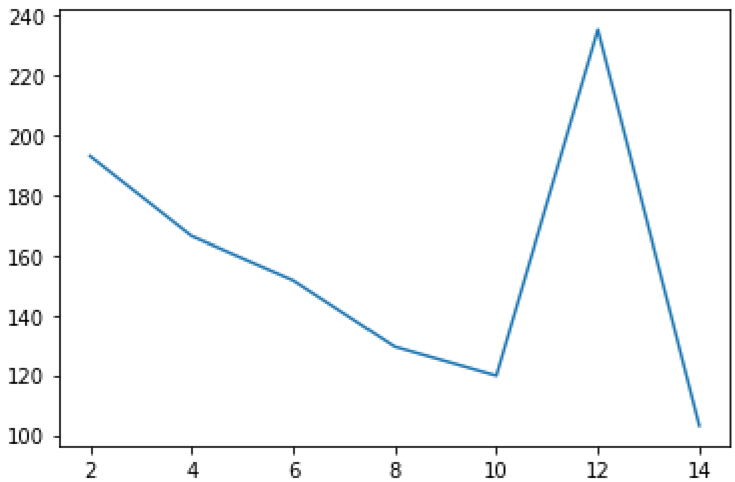
In [8]:
```
def runForKTimes(): #trying to graph the scatter plot not yet complete
    kList=[]
    wccList=[]
    for i in range(2,16,2):
        print("Running K= ",i)
        K,clusters,wcc =kMeansRand(i,dfData_norm,len(dfData))
        kList.append(K)
        wccList.append(wcc)
    return kList,wccList
kList,wccList= runForKTimes()
```

```
Running K=  2
Running K=  4
Running K=  6
Running K=  8
Running K=  10
Running K=  12

C:\Users\johno\Anaconda3\lib\site-packages\ipykernel_launcher.py:26: RuntimeW
arning: invalid value encountered in double_scalars
C:\Users\johno\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: RuntimeW
arning: invalid value encountered in double_scalars
C:\Users\johno\Anaconda3\lib\site-packages\ipykernel_launcher.py:28: RuntimeW
arning: invalid value encountered in double_scalars
C:\Users\johno\Anaconda3\lib\site-packages\ipykernel_launcher.py:29: RuntimeW
arning: invalid value encountered in double_scalars
C:\Users\johno\Anaconda3\lib\site-packages\ipykernel_launcher.py:30: RuntimeW
arning: invalid value encountered in double_scalars
C:\Users\johno\Anaconda3\lib\site-packages\ipykernel_launcher.py:31: RuntimeW
arning: invalid value encountered in double_scalars
C:\Users\johno\Anaconda3\lib\site-packages\ipykernel_launcher.py:32: RuntimeW
arning: invalid value encountered in double_scalars

Running K=  14
```

In [9]:
```
print(kList,wccList)
plt.plot(kList, wccList)
```

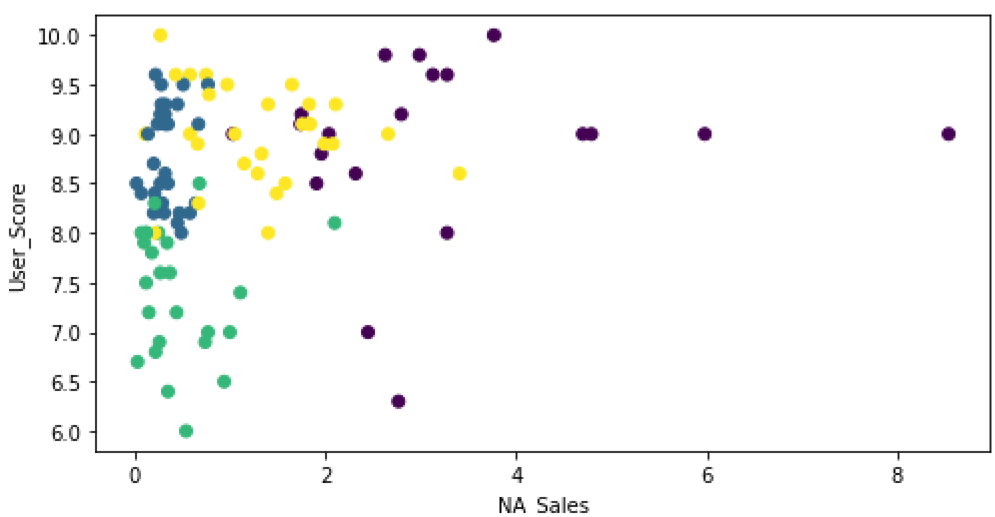
```
[2, 4, 6, 8, 10, 12, 14] [193.1679931046258, 166.47772707051718, 151.76013722
03398, 129.5830255124602, 119.89906462818541, 235.46469454546406, 103.1285814
8685515]
```

Out[9]: [<matplotlib.lines.Line2D at 0x1d6979dfec8>]



In [10]:
```
K,clusters,wcc =kMeansRand(4,dfData_norm,len(dfData))
visualize(clusters,df,"NA_Sales","User_Score")
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 2 2 3 2 1 2 3 3 1 3 2 2 1 3 1 1 1 2 1 1 2 1 2 1 1 2 2 1 3 3 1 1 1 2 1
 3 1 3 3 1 1 1 3 2 2 1 1 2 2 1 1 1 1 1 2 1 2 2 2 2 2 1 2 2]
```



So as of right now I am still struggling with a bug when I am trying to calculate Within Cluster Sum of Squares, so I opted to just use "average distance between each data point and its cluster’s centroid",suggested from class. And I am still getting a bug as the k verse avg distance between point plot does not look right. But I do have initial random clustering completed, so I plotted the cluster with lowest distance on the k plot anyways. But the importance of finding the right k, can not be understated as it completely will affect how we see the data.

Pending Work:

- fix the error in K verse avg distance between point plot
- write hierarchical clustering algo for initialzation for kmeans
- run random initialization 100 times to compare to the hierarchical clustering initialzation kmeans -use sklearn adjusted_rand_score
- create visuals that will include video game names for final clusters, could consider plotting in 3d if time

Aside: If I do not include User_score than the dataframe will have a total of 1308 videogames just with one less column. Do you think it would be more advantagous to have more points or dimensions? Also I get this runtime warningRuntimeWarning: invalid value encountered in double_scalars sometimes and I am not sure why, any glaring errors?