Video Game Sales

Sanay Shah,Ishpreet Kaur,Moheth Muralidharan

12/14/2021

Overview

The video game industry is at the peak of an exciting evolution. These games have been around for decades, providing entertainment for any given age group. They have significantly evolved from the early days of computer games to the latest complex platforms. The Data provided in our project is a combination of 3 datasets based on Video Games Sales.

The data can be described by the Name of the video games, their publishers, the platforms on which they are played, Genre classification, the regional sales mainly in North America, Europe and Japan, the Critic and User Scores etc.

Through this project we have attempted to apply the learnt concepts of Probability, Cluster Analysis, Text Analysis and Time Series Analysis to gain an in-depth understanding of the sales trends undertaken by companies to target its audience.

```
## Loading required package: usethis
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following objects are masked from 'package:lemon':
##
##     CoordCartesian, element_render
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: cluster

## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa

## Loading required package: viridisLite

## NOTE: Either Arial Narrow or Roboto Condensed fonts are required to use these themes.

##       Please use hrbrthemes::import_roboto_condensed() to install Roboto Condensed and

##       if Arial Narrow is not on your system, please see https://bit.ly/arialnarrow

##
## Attaching package: 'tau'

## The following object is masked from 'package:readr':
##
##     tokenize

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

##
## Attaching package: 'nonlinearTseries'

## The following object is masked from 'package:grDevices':
##
##     contourLines

## Rows: 16598 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): Name, Platform, Year, Genre, Publisher
## dbl (6): Rank, NA_Sales, EU_Sales, JP_Sales, Other_Sales, Global_Sales

##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.

## Rows: 18800 Columns: 6

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): name, platform, release_date, summary, user_review
## dbl (1): meta_score

##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.

## Rows: 55792 Columns: 16

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (6): Name, Genre, ESRB_Rating, Platform, Publisher, Developer
## dbl (10): Rank, Critic_Score, User_Score, Total_Shipped, Global_Sales, NA_Sa...

##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

```
summary(df_sales)
```

```
##       Rank           Name             Platform             Year          
##  Min.   :    1   Length:15897       Length:15897       Length:15897      
##  1st Qu.: 4181   Class :character   Class :character   Class :character  
##  Median : 8312   Mode  :character   Mode  :character   Mode  :character  
##  Mean   : 8319                                                           
##  3rd Qu.:12465                                                           
##  Max.   :16600                                                           
##                                                                          
##     Genre            Publisher            NA_Sales          EU_Sales      
##  Length:15897       Length:15897       Min.   : 0.0000   Min.   : 0.0000  
##  Class :character   Class :character   1st Qu.: 0.0000   1st Qu.: 0.0000  
##  Mode  :character   Mode  :character   Median : 0.0800   Median : 0.0200  
##                                        Mean   : 0.2634   Mean   : 0.1433  
##                                        3rd Qu.: 0.2400   3rd Qu.: 0.1100  
##                                        Max.   :41.4900   Max.   :29.0200  
##                                                                          
##     JP_Sales        Other_Sales        Global_Sales        Rating         
##  Min.   :0.00000   Min.   : 0.00000   Min.   : 0.0100   Length:15897      
##  1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.: 0.0600   Class :character
##  Median :0.00000   Median : 0.01000   Median : 0.1700   Mode  :character
##  Mean   :0.07142   Mean   : 0.04736   Mean   : 0.5257
##  3rd Qu.:0.03000   3rd Qu.: 0.03000   3rd Qu.: 0.4700
##  Max.   :6.81000   Max.   :10.57000   Max.   :82.7400
##
##   Developer          Critic_Score      User_Score
##  Length:15897       Min.   : 1.000   Min.   : 2.000
##  Class :character   1st Qu.: 6.500   1st Qu.: 8.000
##  Mode  :character   Median : 7.500   Median : 8.800
##                     Mean   : 7.235   Mean   : 8.518
##                     3rd Qu.: 8.300   3rd Qu.: 9.300
##                     Max.   :10.000   Max.   :10.000
##                     NA’s   :11716    NA’s   :15713
```

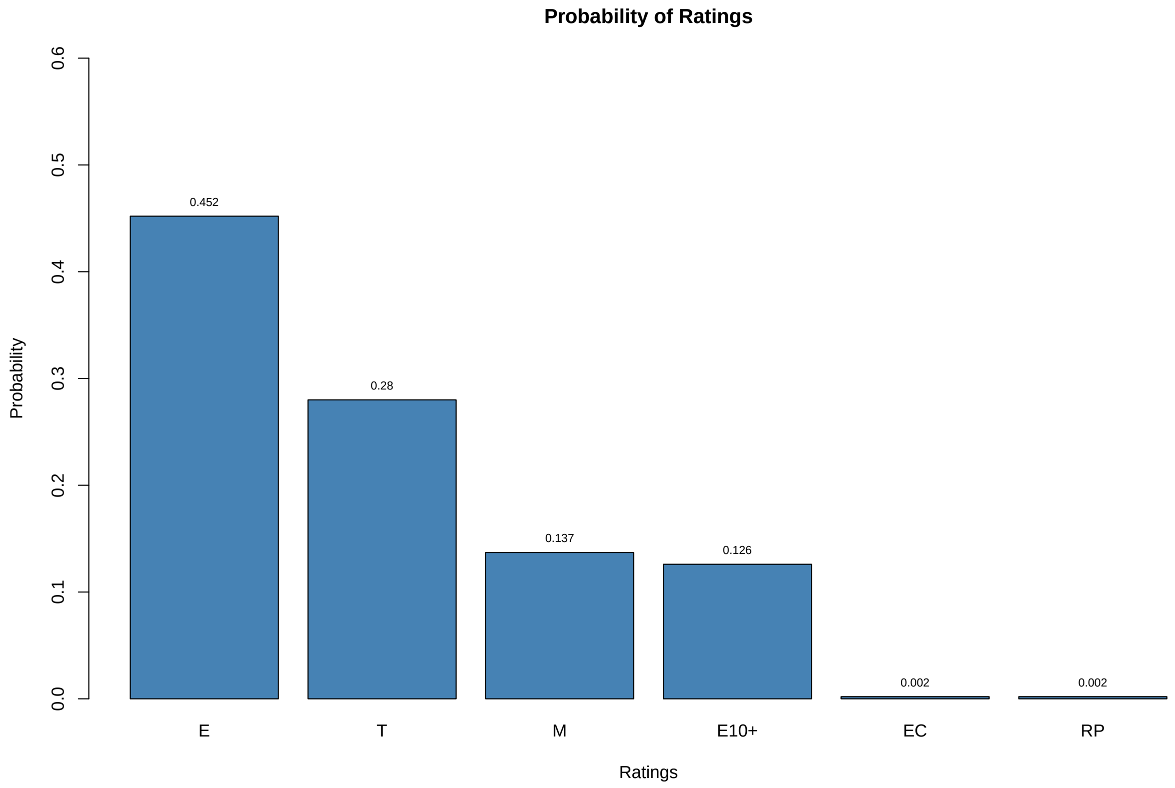
```
summary(df_games)
```

```
##       Rank          Name             Platform             Year
##  Min.   :    1   Length:1921        Length:1921        Length:1921
##  1st Qu.: 3397   Class :character   Class :character   Class :character
##  Median : 7684   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 7876
##  3rd Qu.:12398
##  Max.   :16542
##
##     Genre            Publisher            NA_Sales          EU_Sales
##  Length:1921        Length:1921        Min.   : 0.0000   Min.   : 0.0000
##  Class :character   Class :character   1st Qu.: 0.0100   1st Qu.: 0.0100
##  Mode  :character   Mode  :character   Median : 0.1000   Median : 0.0300
##                                        Mean   : 0.3477   Mean   : 0.2459
##                                        3rd Qu.: 0.2800   3rd Qu.: 0.1600
##                                        Max.   :41.4900   Max.   :29.0200
##
##     JP_Sales        Other_Sales       Global_Sales        Rating
##  Min.   :0.00000   Min.   :0.00000   Min.   : 0.0100   Length:1921
##  1st Qu.:0.00000   1st Qu.:0.01000   1st Qu.: 0.0600   Class :character
##  Median :0.00000   Median :0.02000   Median : 0.2000   Mode  :character
##  Mean   :0.09828   Mean   :0.07323   Mean   : 0.7655
##  3rd Qu.:0.01000   3rd Qu.:0.05000   3rd Qu.: 0.5900
##  Max.   :6.50000   Max.   :8.46000   Max.   :82.7400
##
##   Developer          Critic_Score   Release_Date         Summary
##  Length:1921        Min.   :2.000   Length:1921        Length:1921
##  Class :character   1st Qu.:6.900   Class :character   Class :character
##  Mode  :character   Median :7.500   Mode  :character   Mode  :character
##                     Mean   :7.487
##                     3rd Qu.:8.200
##                     Max.   :9.700
##                     NA’s   :811
##    Metascore      User_Score
##  Min.   :23.00   Length:1921
##  1st Qu.:65.00   Class :character
##  Median :73.00   Mode  :character
##  Mean   :72.15
##  3rd Qu.:80.00
##  Max.   :97.00
##
```

** Probability **

** Q.1 What are the games developed based on different age groups? **

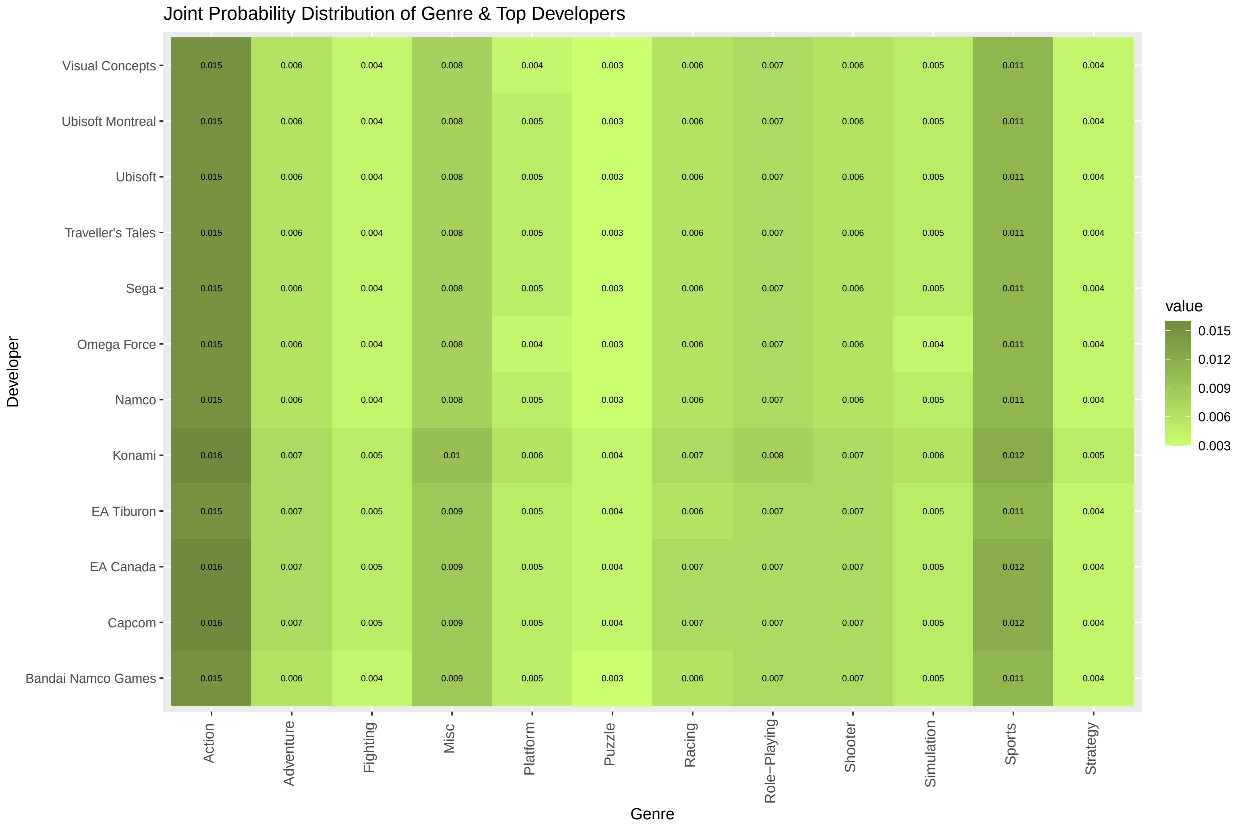
total_records
11994



Conclusion: Based on the rating column we can figure out the probability of the number of games developed in a particular age group, where E=Everyone, T=Teen, M=Mature, E10+= Everyone10+, EC= Early childhood, RP= Rating Pending.

We can see that the most games are for 'Everyone' followed by 'Teens', 'Mature', 'Everyone10+'. Through this we can conclude that the games are targeted for larger audience in order to earn more Revenue.

Q.2 Top Developers prefered to build games in which Genre?

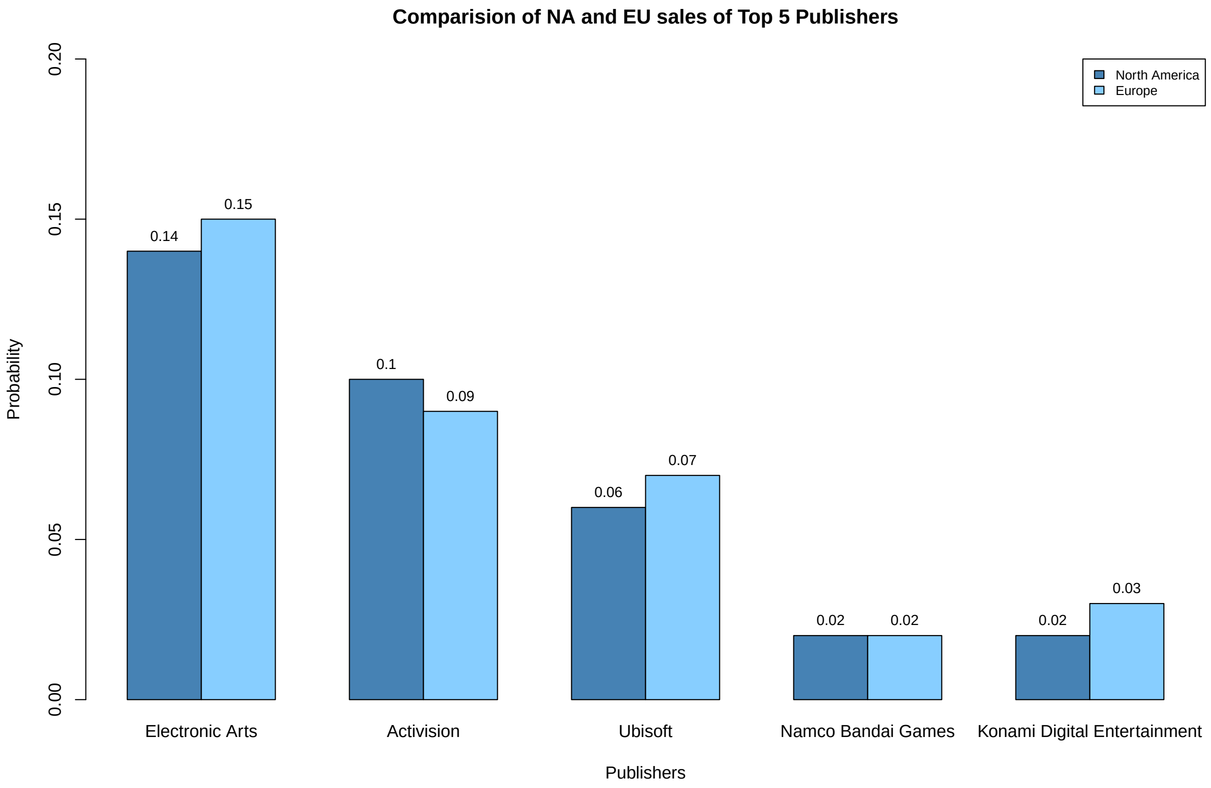


Joint Probability Distribution of Genre & Top Developers

** Conclusion: ** The analysis measures the likelihood for the Top 12 Developers to develop a game for a specific Genre using Joint Probabilty Distribution.

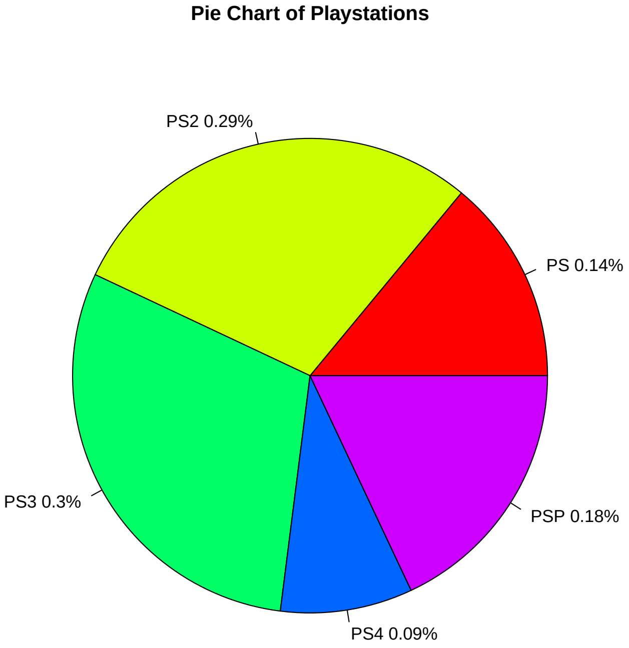
##From the above graph we can conclude that the developers prefer to develop games of Action and Sports Genre the most as they have the highest Joint Probability.

** Q.3 Find the comparison between the sales of North America and Europe on the basis of the Top 5 Publishers **



Conclusion: The North America and Europe sales of the Top 5 publishers have been compared by using probability. We can find that the sales for Electronic Arts is more in the Europe region compared to the North American region while for activision, the sales in Europe is more compared to the North American sales.

**** Q.4 Distribution of Video Games sales for the PS Series****

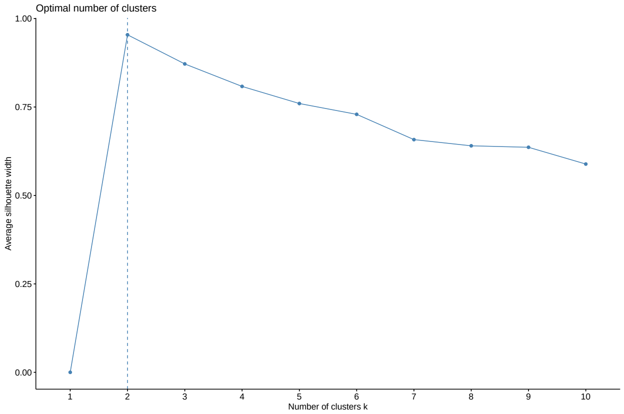


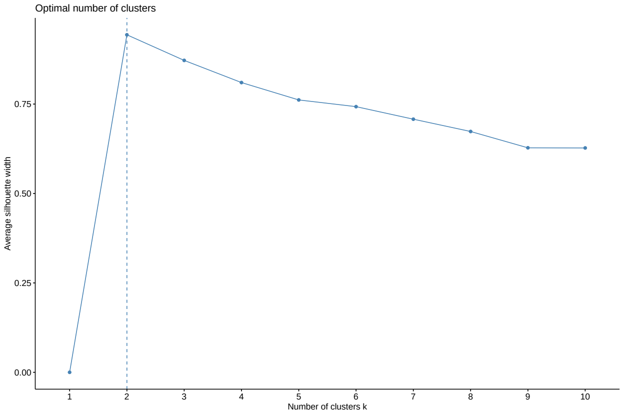
Conclusion: The games produced for a particular Playstation platform is found using Probability Distribution Function. The probability of the games produced for a particular Platform among the Playstation series can be seen from the graph.

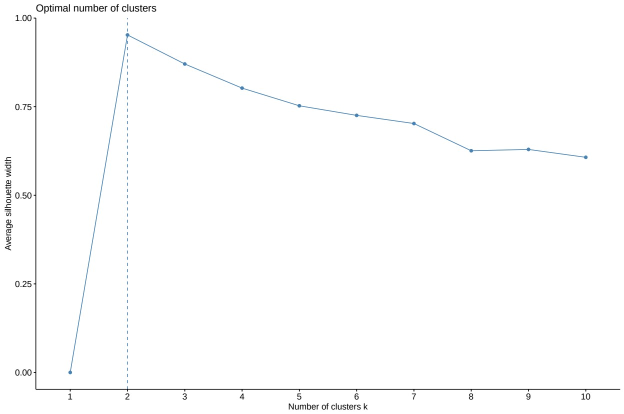
##Having first transitioned to games in the digital space, now expanding to gaming across platforms and devices we can conclude that there were around 60% sales of the video games played on PS2 and PS3 than any other PS Station platform. We can say that these two platforms were the most popular amongst all.

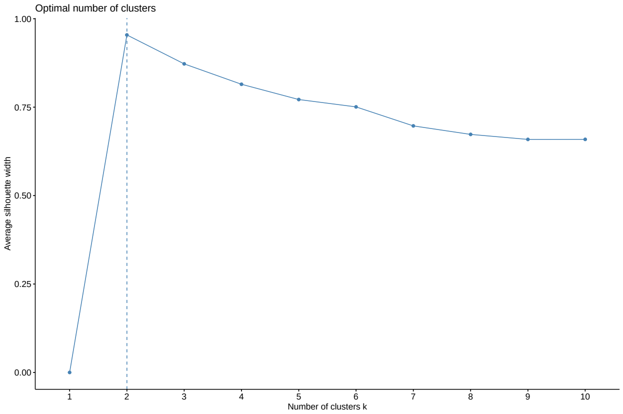
Clustering Analysis

** Q.5 Classify each region-wise sales with respect to the Global sales**









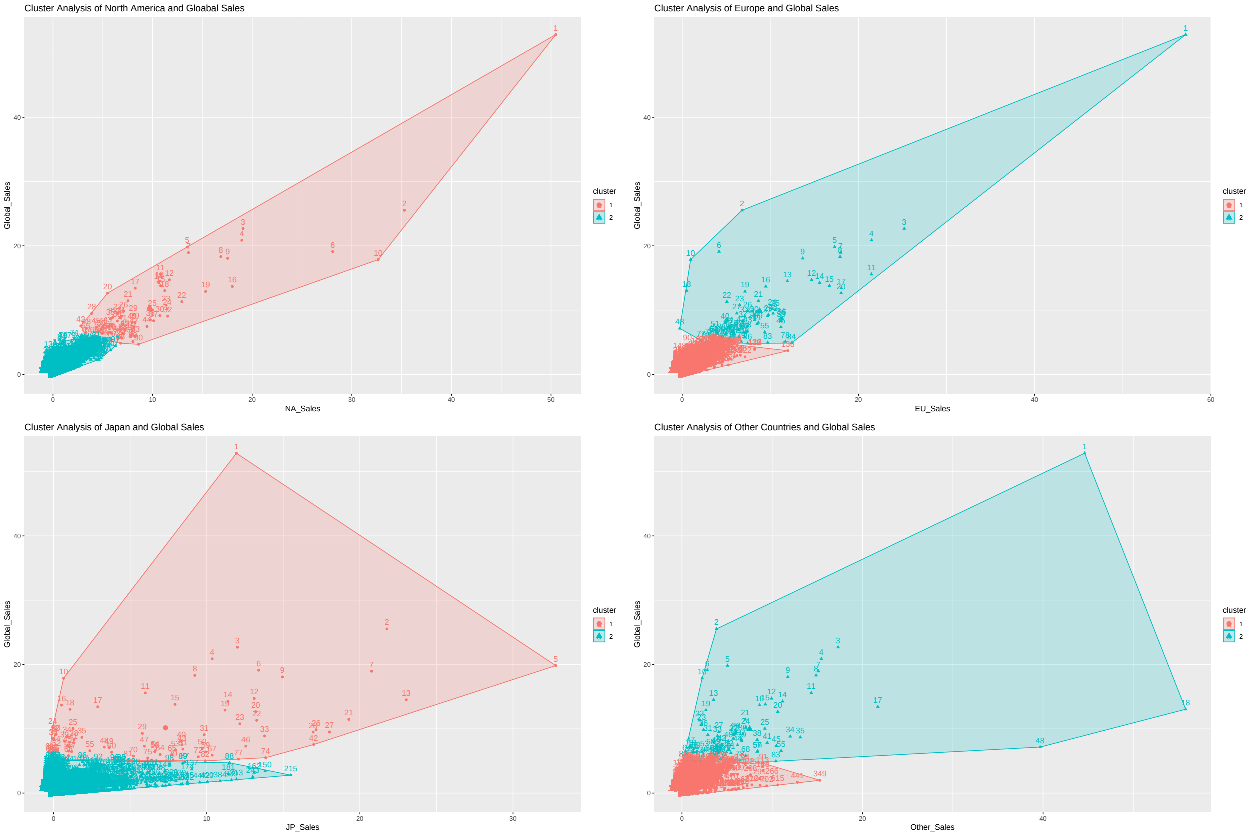
## **Conclusion:** By using silhouette method in k-means we could observe that the optimal number of clusters are 2. We also observed that after reaching the optimal value the average silhouette width keeps decreasing gradually.

```
##    NA_Sales Global_Sales
## 1 8.1941026   16.4969231
## 2 0.2272282    0.4620872
```

```
##    EU_Sales Global_Sales
## 1 0.1237788    0.4586441
## 2 4.5902353   15.8452941
```

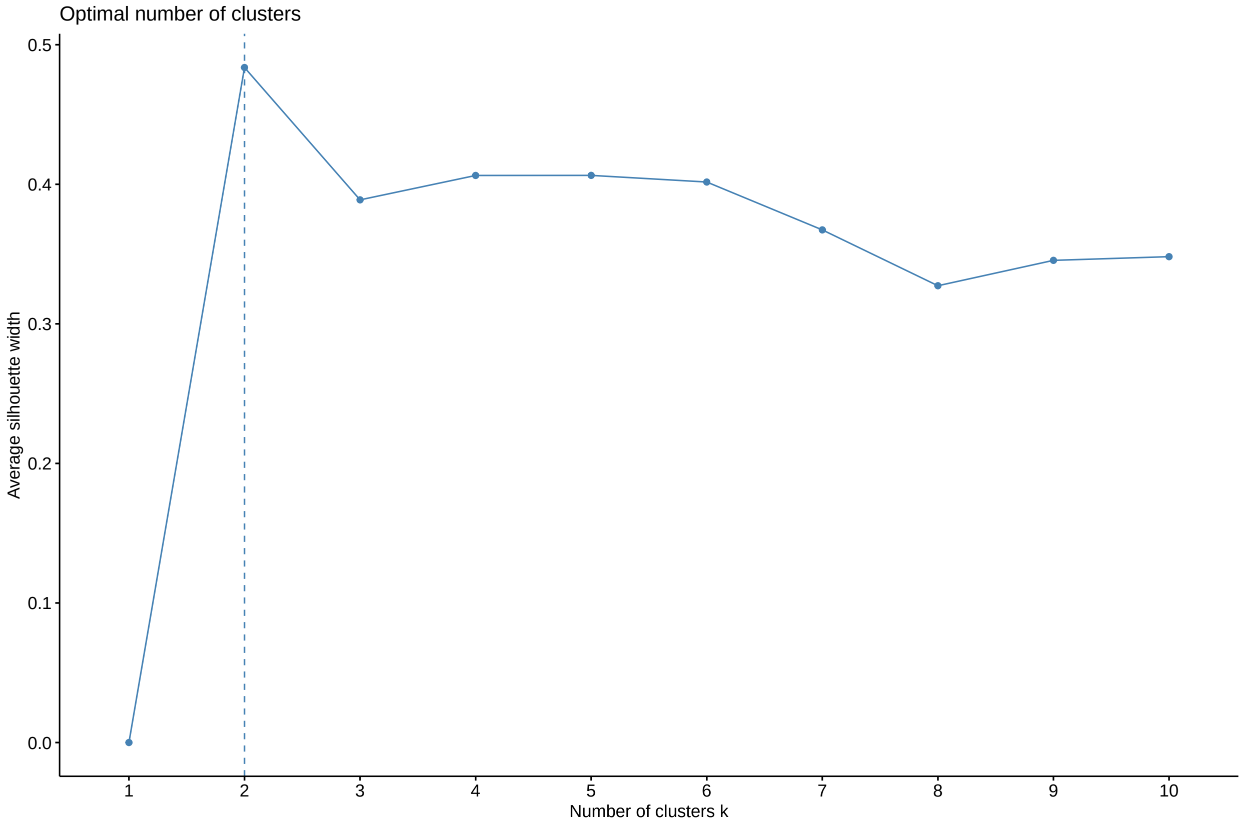
```
##     JP_Sales Global_Sales
## 1 2.33775000    16.318250
## 2 0.06683618     0.461011
```

```
##   Other_Sales Global_Sales
## 1  0.04095671    0.4595677
## 2  1.46204819   16.0322892
```



## **Conclusion** By performing Cluster analysis, for region-wise sales with respect to the Gloabl sales we can observe the similarities and dissimilaries in video game sales trend for respective regions. Companies can make better, data-driven decisions by identifying the pattern of sales in each region

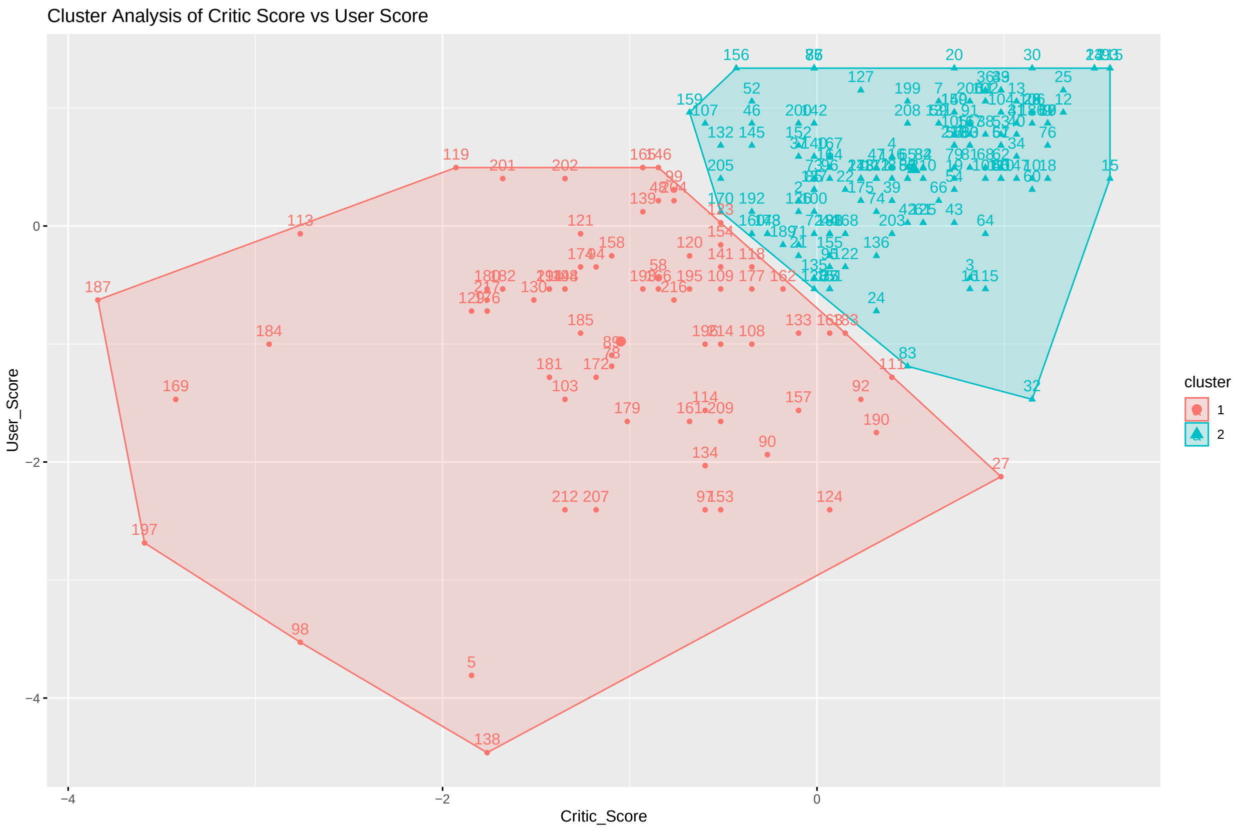
** Q.6 Cluster Analysis based on Critic Scores and User Scores **



```
km.criticUser$centers
```

```
##   Critic_Score User_Score
## 1     6.858333   7.525000
## 2     8.739726   9.086301
```

```
km.CriticUserCluster
```



Conclusion:

The kmeans function outputs the results of the clustering. We can observe the following:-

a. cluster means- the centroid vector values under Critic Score and User Score columns

b. clustering vector- the group in which each observation was allocated that is in groups of 1 and

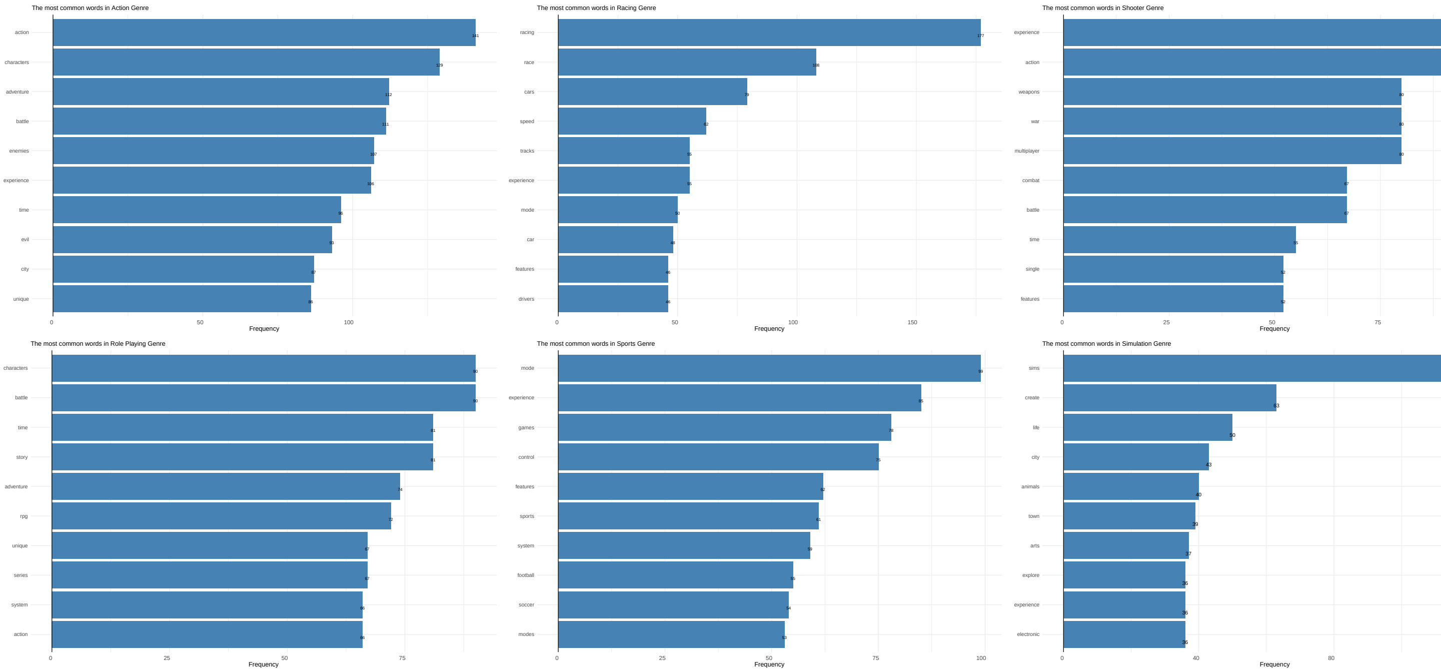
We performed cluster analysis on Critic Scores and User Scores, together to analyze consumer purchase trends. We can observe that the User Score and Critic Scores are similar, they go hand in hand. Concluding that the Users Score for a particular video game is largely made based on the critic scores

Text Mining

Q.7 Based on the Description of each game, how can they be categorized into different Genres?

```
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
```

```
## Selecting by n
## Selecting by n
## Selecting by n
## Selecting by n
## Selecting by n
## Selecting by n
```



Conclusion:

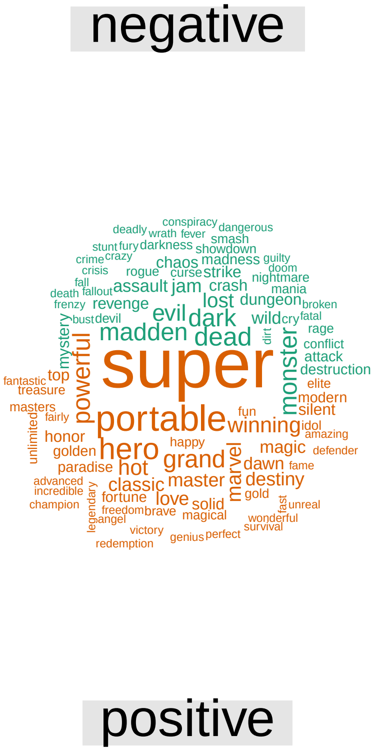
The above analysis help us to identify the most frequently used words in a particular Genre,and categorize them for customers to easily pick their preferred games.

For instance, we can conclude from the graph that for 'Racing' words like cars, speed, tracks, drivers are few of the most frequently used words.

**** Q.8 Sentiment Analysis based on the Name of the Games and how do they play a role to target customers.****

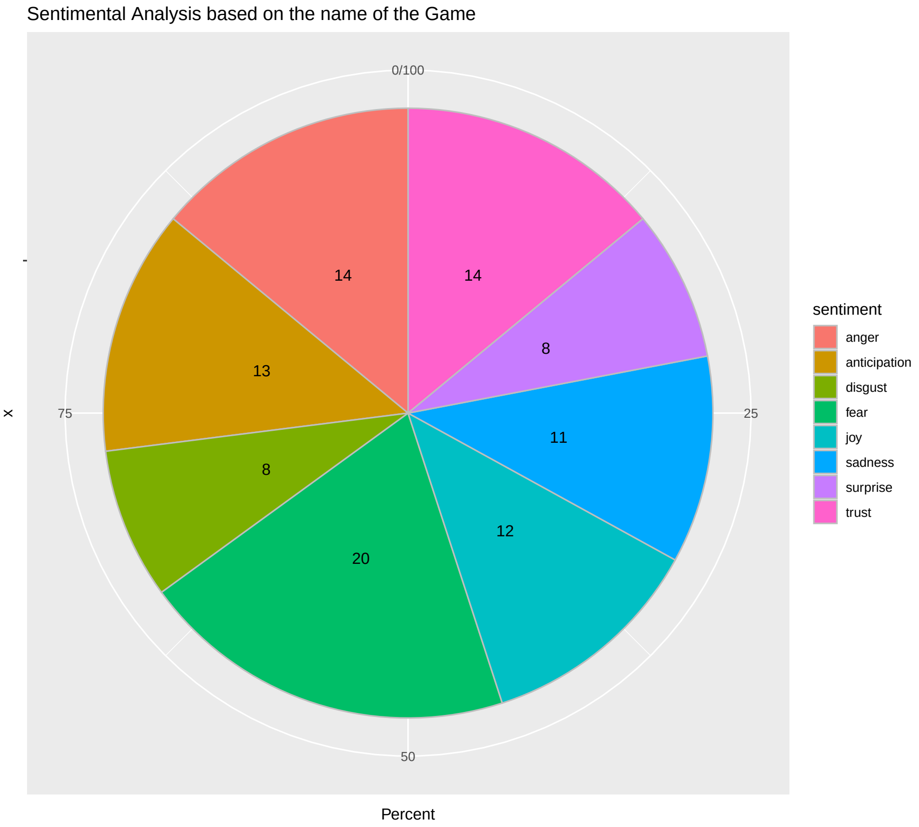
```
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
## Joining, by = "word"
```

```
## Joining, by = "word"
```



```
## Joining, by = "word"
```

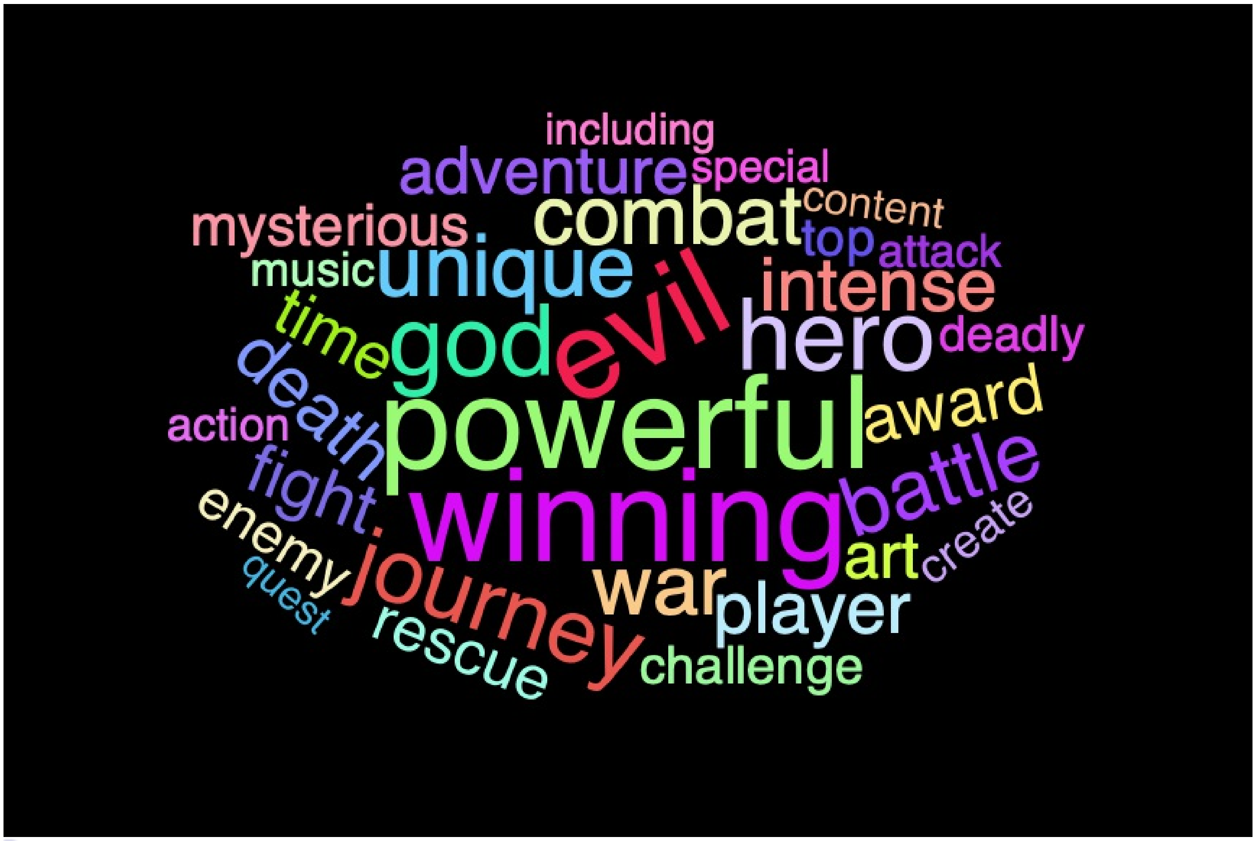
```
## NULL
```



## **Conclusion:** ## Sentiment Analysis were performed on the name of the Games, which were divided into Positive and Negative sentiments and furthermore, they were categorzised into Emotional Sentiments like Anger, Fear, Disgust, Joy, Surprise etc. ## hrough this we can understand how publishers strategise Words with emotions while naming the Games, which attracts the customers to purchase based on their prefered Emotions. ## Based on these analysis we can say that 20% of the words used were for fear, followed by anger and trust at 14% each. This is how the publishers strategise their sales based on emotions to lure the customers.

** Q.9 Sentiment Analysis based on Top 100 Metascores **

```
## Joining, by = "word"
## Joining, by = "word"
```



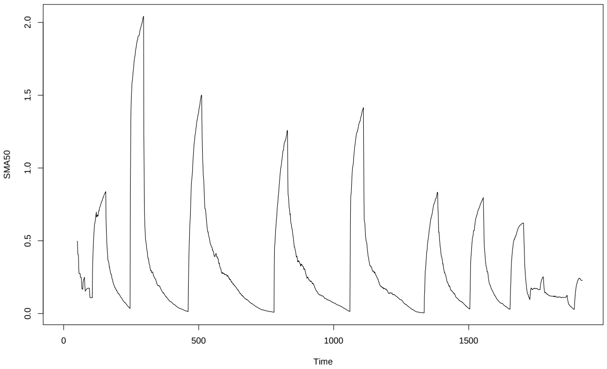
We used Text Mining on Names of the games to analyze the data for Sentiment Analysis.

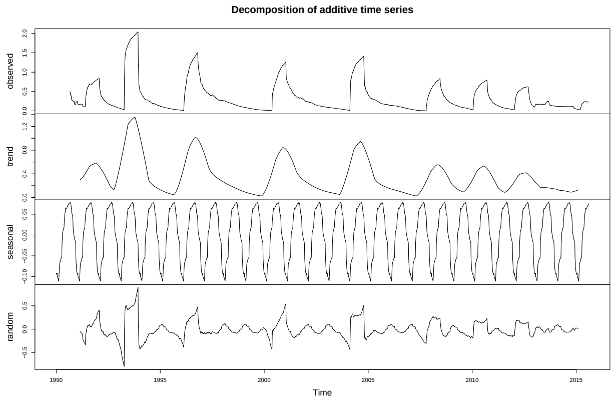
It's often when consumer purchase products on the basis of the product summary/ description.

##Here, we can see the words used repeatedly in description of the top 100 video games, to understand the consumer trends on the basis of usage of words that were attracting consumers to purchase the game.

Time Series

** Q.10 What would be the trend for the next 50 days of the sales in North America ?**

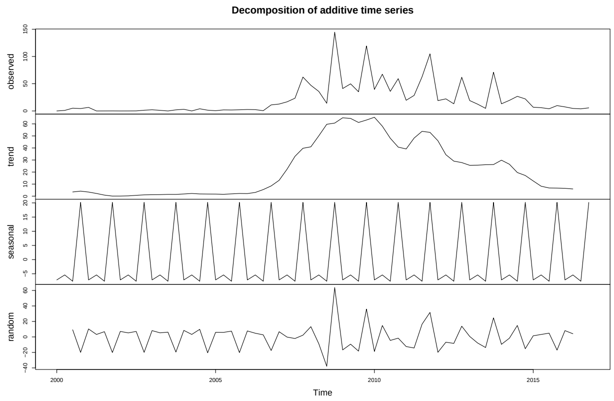


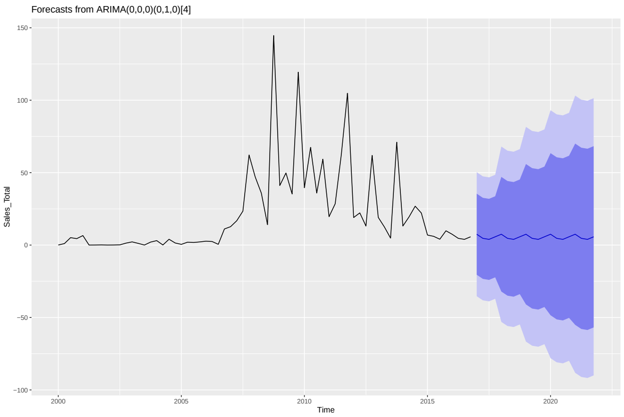


## **Conclusion :** The 50-day simple moving average (SMA) is used by traders as an effective trend indicator. The 50-day average is considered the most important because it’s the first line of support in an uptrend or the first line of resistance in a downtrend. #From the Decomposition graph we can say that the sales are pretty low and would gradually increase near 40th or 45th day.

** Q.11 What would be the global sales for the next 5 years?**

```
## ‘summarise()‘ has grouped output by ’Year’. You can override using the ‘.groups‘ argument.
```

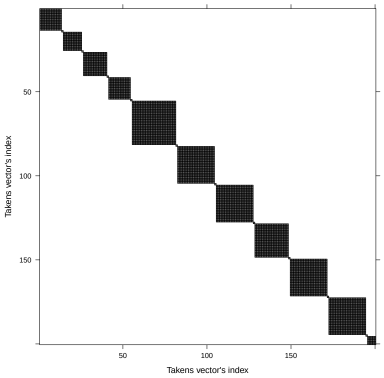




# ** Conclusion : **Using Decomposition of additive time series we are Decomposing global sales.The seasonal ,trend and the observed valuesare components of additive time series.Further we are using Arima model -(p), the number of lag observations or autoregressive terms in the model; I (d), the difference in the nonseasonal observations; and MA (q), the size of the moving average window .Arima(p,d,q)->(0,1,0)is best fitted for our analysis for prediction of Global sales for the next 5 years.The dark line is 95 % accurate We observe that there is no significant growth in the sales of the video games.As shades change (become lighter) the accuracy reduces by 10 %.

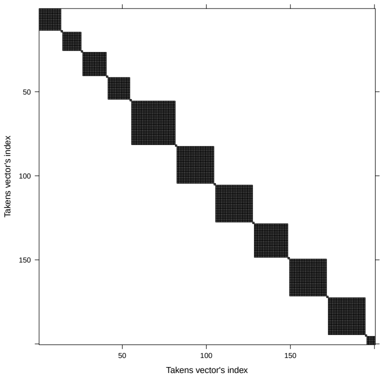
** Q.12 Reccurence Plot for Global Sales **

Embedding Dim=2,Time Lag=1,Radius=0.01



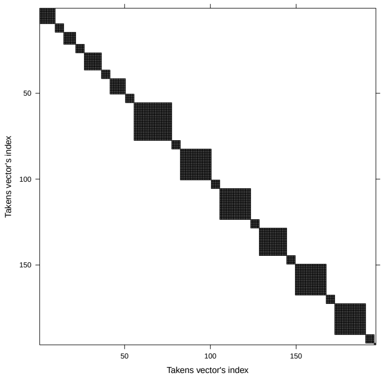
Dimensions: 200 x 200

Embedding Dim=2,Time Lag=1,Radius=0.01



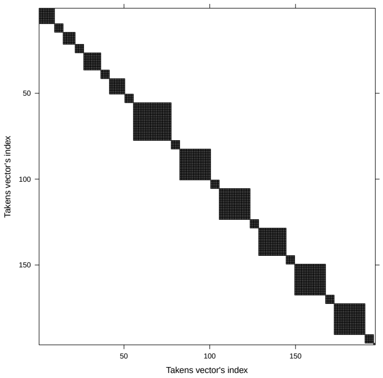
Dimensions: 200 x 200

Embedding Dim=2,Time Lag=5,Radius=0.01



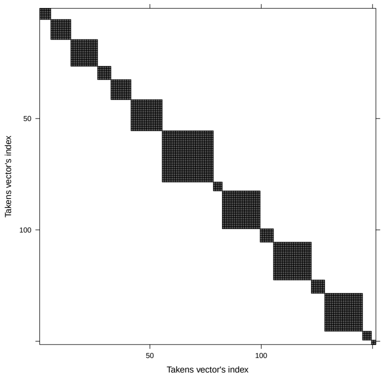
Dimensions: 196 x 196

Embedding Dim=2,Time Lag=5,Radius=0.01



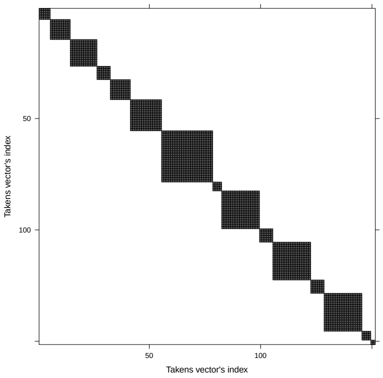
Dimensions: 196 x 196

Embedding Dim=2,Time Lag=50,Radius=0.01

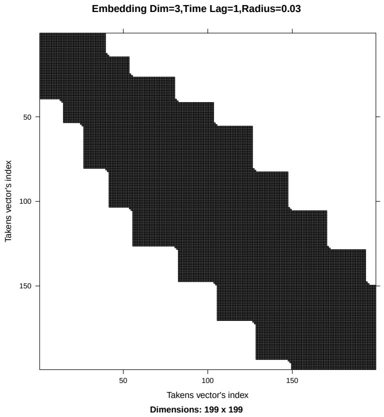


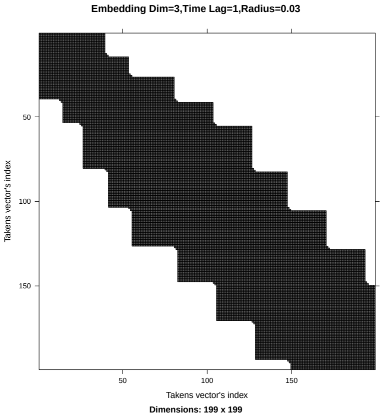
Dimensions: 151 x 151

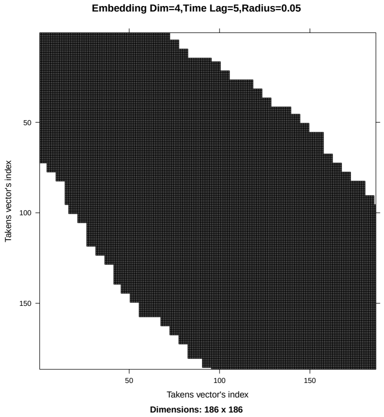
Embedding Dim=2,Time Lag=50,Radius=0.01

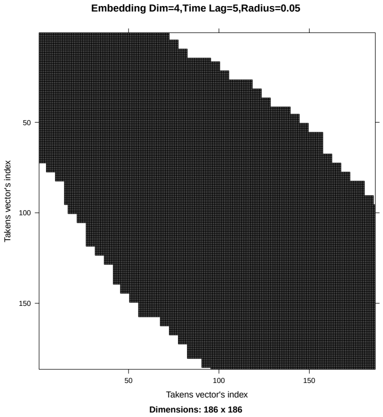


Dimensions: 151 x 151





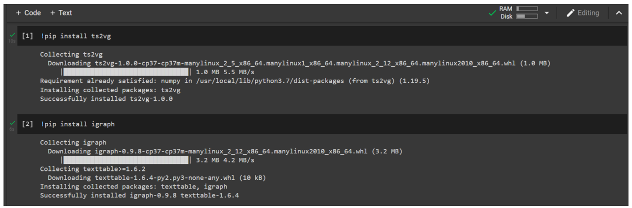




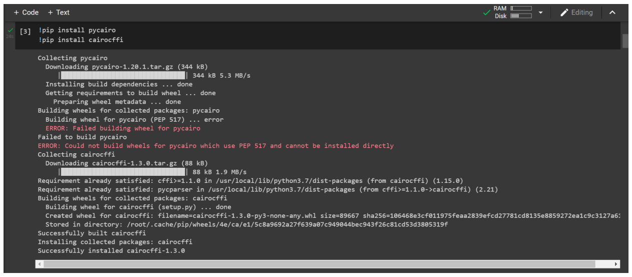
** Conclusion :** Through RQA we can quantify the sales and duration of the recurrences in the phase space.

##We can observe how the graph changes when we change the Lag it becomes thinner and while increasing the dimension it becomes thicker as also the radius is increased.

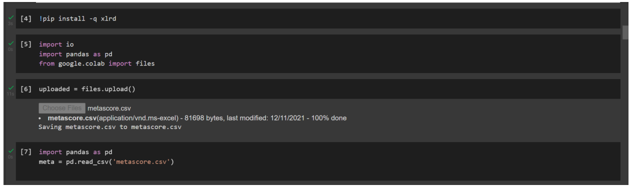
** Q.13 Visibility Graphs and the Horizontal Visibility Graph of the Metascore using Python **



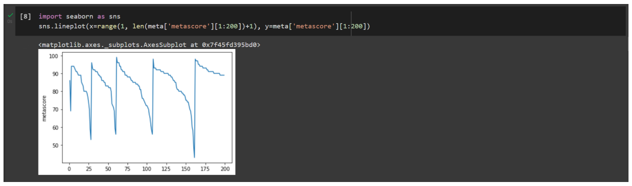
```
[1] !pip install ts2vg

    Collecting ts2vg
      Downloading ts2vg-1.0.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.0 MB)
         |████████████████████████████████| 1.0 MB 5.5 MB/s
    Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from ts2vg) (1.19.5)
    Installing collected packages: ts2vg
    Successfully installed ts2vg-1.0.0

[2] !pip install igraph

    Collecting igraph
      Downloading igraph-0.9.8-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.2 MB)
         |████████████████████████████████| 3.2 MB 4.2 MB/s
    Collecting texttable>=1.6.2
      Downloading texttable-1.6.4-py2.py3-none-any.whl (10 kB)
    Installing collected packages: texttable, igraph
    Successfully installed igraph-0.9.8 texttable-1.6.4
```



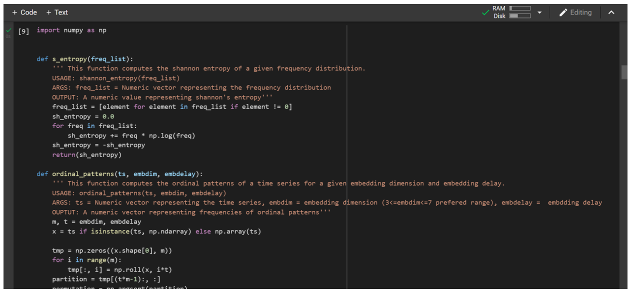
```
[3] !pip install pycairo
    !pip install cairocffi

    Collecting pycairo
      Downloading pycairo-1.20.1.tar.gz (344 kB)
         |████████████████████████████████| 344 kB 5.3 MB/s
      Installing build dependencies ... done
      Getting requirements to build wheel ... done
        Preparing wheel metadata ... done
    Building wheels for collected packages: pycairo
      Building wheel for pycairo (PEP 517) ... error
      ERROR: Failed building wheel for pycairo
    Failed to build pycairo
    ERROR: Could not build wheels for pycairo which use PEP 517 and cannot be installed directly
    Collecting cairocffi
      Downloading cairocffi-1.3.0.tar.gz (88 kB)
         |████████████████████████████████| 88 kB 1.9 MB/s
    Requirement already satisfied: cffi>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from cairocffi) (1.15.0)
    Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from cffi>=1.1.0->cairocffi) (2.21)
    Building wheels for collected packages: cairocffi
      Building wheel for cairocffi (setup.py) ... done
      Created wheel for cairocffi: filename=cairocffi-1.3.0-py3-none-any.whl size=89667 sha256=106468e3cf011975feaa2839efcd27781cd8135e8859272ea1c9c3127a6
      Stored in directory: /root/.cache/pip/wheels/4e/ca/e1/5c8a9692a27f639a07c949044bec943f26c81cd53d3805319f
    Successfully built cairocffi
    Installing collected packages: cairocffi
    Successfully installed cairocffi-1.3.0
```



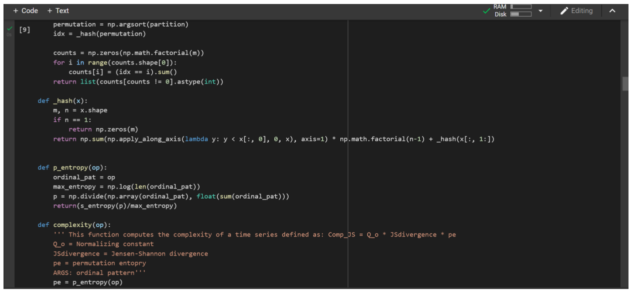
```
[4] !pip install -q xlrd

[5] import io
    import pandas as pd
    from google.colab import files

[6] uploaded = files.upload()

    Choose Files metascore.csv
    • metascore.csv(application/vnd.ms-excel) - 81698 bytes, last modified: 12/11/2021 - 100% done
    Saving metascore.csv to metascore.csv

[7] import pandas as pd
    meta = pd.read_csv('metascore.csv')
```



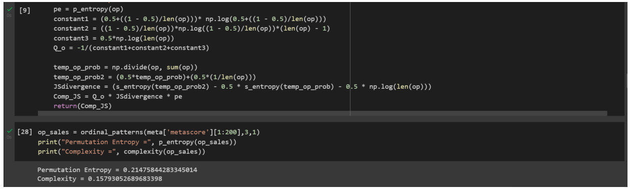
```
[8] import seaborn as sns
    sns.lineplot(x=range(1, len(meta['metascore'][1:200])+1), y=meta['metascore'][1:200])

    <matplotlib.axes._subplots.AxesSubplot at 0x7f45fd395bd0>
```



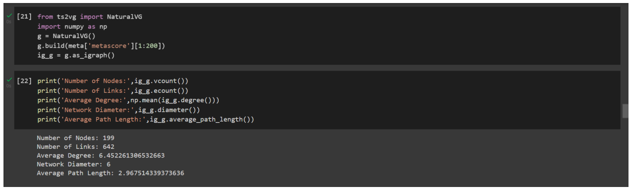
```
[9] import numpy as np


    def s_entropy(freq_list):
        ''' This function computes the shannon entropy of a given frequency distribution.
        USAGE: shannon_entropy(freq_list)
        ARGS: freq_list = Numeric vector representing the frequency distribution
        OUTPUT: A numeric value representing shannon's entropy'''
        freq_list = [element for element in freq_list if element != 0]
        sh_entropy = 0.0
        for freq in freq_list:
            sh_entropy += freq * np.log(freq)
        sh_entropy = -sh_entropy
        return(sh_entropy)

    def ordinal_patterns(ts, embdim, embdelay):
        ''' This function computes the ordinal patterns of a time series for a given embedding dimension and embedding delay.
        USAGE: ordinal_patterns(ts, embdim, embdelay)
        ARGS: ts = Numeric vector representing the time series, embdim = embedding dimension (3<=embdim<=7 prefered range), embdelay =  embdding delay
        OUPTUT: A numeric vector representing frequencies of ordinal patterns'''
        m, t = embdim, embdelay
        x = ts if isinstance(ts, np.ndarray) else np.array(ts)

        tmp = np.zeros((x.shape[0], m))
        for i in range(m):
            tmp[:, i] = np.roll(x, i*t)
        partition = tmp[(t*m-1):, :]
```



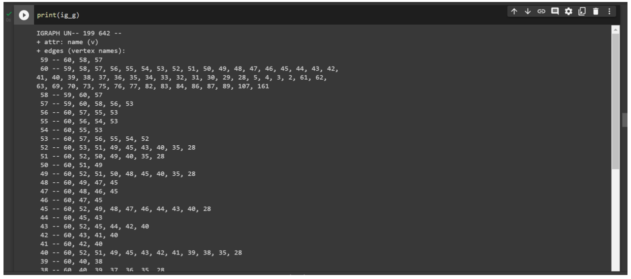
```
[9]     permutation = np.argsort(partition)
        idx = _hash(permutation)

        counts = np.zeros(np.math.factorial(m))
        for i in range(counts.shape[0]):
            counts[i] = (idx == i).sum()
        return list(counts[counts != 0].astype(int))

    def _hash(x):
        m, n = x.shape
        if n == 1:
            return np.zeros(m)
        return np.sum(np.apply_along_axis(lambda y: y < x[:, 0], 0, x), axis=1) * np.math.factorial(n-1) + _hash(x[:, 1:])


    def p_entropy(op):
        ordinal_pat = op
        max_entropy = np.log(len(ordinal_pat))
        p = np.divide(np.array(ordinal_pat), float(sum(ordinal_pat)))
        return(s_entropy(p)/max_entropy)

    def complexity(op):
        ''' This function computes the complexity of a time series defined as: Comp_JS = Q_o * JSdivergence * pe
        Q_o = Normalizing constant
        JSdivergence = Jensen-Shannon divergence
        pe = permutation entopry
        ARGS: ordinal pattern'''
        pe = p_entropy(op)
```



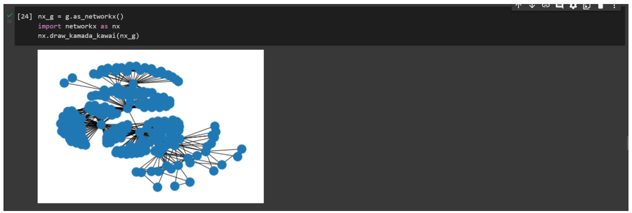
```
[9]     pe = p_entropy(op)
        constant1 = (0.5+((1 - 0.5)/len(op)))* np.log(0.5+((1 - 0.5)/len(op)))
        constant2 = ((1 - 0.5)/len(op))*np.log((1 - 0.5)/len(op))*(len(op) - 1)
        constant3 = 0.5*np.log(len(op))
        Q_o = -1/(constant1+constant2+constant3)

        temp_op_prob = np.divide(op, sum(op))
        temp_op_prob2 = (0.5*temp_op_prob)+(0.5*(1/len(op)))
        JSdivergence = (s_entropy(temp_op_prob2) - 0.5 * s_entropy(temp_op_prob) - 0.5 * np.log(len(op)))
        Comp_JS = Q_o * JSdivergence * pe
        return(Comp_JS)

[28] op_sales = ordinal_patterns(meta['metascore'][1:200],3,1)
     print("Permutation Entropy =", p_entropy(op_sales))
     print("Complexity =", complexity(op_sales))

     Permutation Entropy = 0.21475844283345014
     Complexity = 0.15793052689683398
```



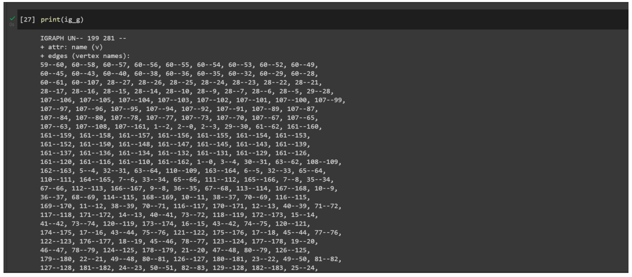
```
[21] from ts2vg import NaturalVG
     import numpy as np
     g = NaturalVG()
     g.build(meta['metascore'][1:200])
     ig_g = g.as_igraph()

[22] print('Number of Nodes:',ig_g.vcount())
     print('Number of Links:',ig_g.ecount())
     print('Average Degree:',np.mean(ig_g.degree()))
     print('Network Diameter:',ig_g.diameter())
     print('Average Path Length:',ig_g.average_path_length())

     Number of Nodes: 199
     Number of Links: 642
     Average Degree: 6.452261306532663
     Network Diameter: 6
     Average Path Length: 2.967514339373636
```



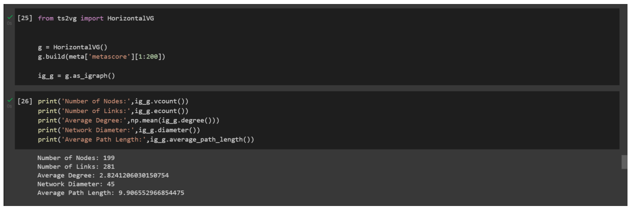
```
print(ig_g)

IGRAPH UN-- 199 642 --
+ attr: name (v)
+ edges (vertex names):
 59 -- 60, 58, 57
 60 -- 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 42,
41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 5, 4, 3, 2, 61, 62,
63, 69, 70, 73, 75, 76, 77, 82, 83, 84, 86, 87, 89, 107, 161
 58 -- 59, 60, 57
 57 -- 59, 60, 58, 56, 53
 56 -- 60, 57, 55, 53
 55 -- 60, 56, 54, 53
 54 -- 60, 55, 53
 53 -- 60, 57, 56, 55, 54, 52
 52 -- 60, 53, 51, 49, 45, 43, 40, 35, 28
 51 -- 60, 52, 50, 49, 40, 35, 28
 50 -- 60, 51, 49
 49 -- 60, 52, 51, 50, 48, 45, 40, 35, 28
 48 -- 60, 49, 47, 45
 47 -- 60, 48, 46, 45
 46 -- 60, 47, 45
 45 -- 60, 52, 49, 48, 47, 46, 44, 43, 40, 28
 44 -- 60, 45, 43
 43 -- 60, 52, 45, 44, 42, 40
 42 -- 60, 43, 41, 40
 41 -- 60, 42, 40
 40 -- 60, 52, 51, 49, 45, 43, 42, 41, 39, 38, 35, 28
 39 -- 60, 40, 38
 38 -- 60, 40, 39, 37, 36, 35, 28
```



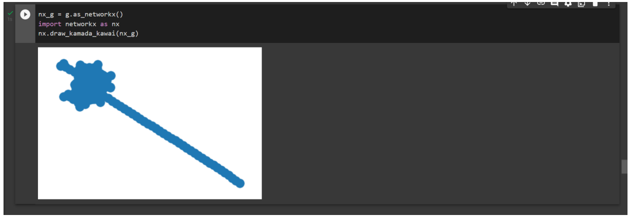
```
[24] nx_g = g.as_networkx()
     import networkx as nx
     nx.draw_kamada_kawai(nx_g)
```



```
[27] print(ig_g)

     IGRAPH UN-- 199 281 --
     + attr: name (v)
     + edges (vertex names):
     59--60, 60--58, 60--57, 60--56, 60--55, 60--54, 60--53, 60--52, 60--49,
     60--45, 60--43, 60--40, 60--38, 60--36, 60--35, 60--32, 60--29, 60--28,
     60--61, 60--107, 28--27, 28--26, 28--25, 28--24, 28--23, 28--22, 28--21,
     28--17, 28--16, 28--15, 28--14, 28--10, 28--9, 28--7, 28--6, 28--5, 29--28,
     107--106, 107--105, 107--104, 107--103, 107--102, 107--101, 107--100, 107--99,
     107--97, 107--96, 107--95, 107--94, 107--92, 107--91, 107--89, 107--87,
     107--84, 107--80, 107--78, 107--77, 107--73, 107--70, 107--67, 107--65,
     107--63, 107--108, 107--161, 1--2, 2--0, 2--3, 29--30, 61--62, 161--160,
     161--159, 161--158, 161--157, 161--156, 161--155, 161--154, 161--153,
     161--152, 161--150, 161--148, 161--147, 161--145, 161--143, 161--139,
     161--137, 161--136, 161--134, 161--132, 161--131, 161--129, 161--126,
     161--120, 161--116, 161--110, 161--162, 1--0, 3--4, 30--31, 63--62, 108--109,
     162--163, 5--4, 32--31, 63--64, 110--109, 163--164, 6--5, 32--33, 65--64,
     110--111, 164--165, 7--6, 33--34, 65--66, 111--112, 165--166, 7--8, 35--34,
     67--66, 112--113, 166--167, 9--8, 36--35, 67--68, 113--114, 167--168, 10--9,
     36--37, 68--69, 114--115, 168--169, 10--11, 38--37, 70--69, 116--115,
     169--170, 11--12, 38--39, 70--71, 116--117, 170--171, 12--13, 40--39, 71--72,
     117--118, 171--172, 14--13, 40--41, 73--72, 118--119, 172--173, 15--14,
     41--42, 73--74, 120--119, 173--174, 16--15, 43--42, 74--75, 120--121,
     174--175, 17--16, 43--44, 75--76, 121--122, 175--176, 17--18, 45--44, 77--76,
     122--123, 176--177, 18--19, 45--46, 78--77, 123--124, 177--178, 19--20,
     46--47, 78--79, 124--125, 178--179, 21--20, 47--48, 80--79, 126--125,
     179--180, 22--21, 49--48, 80--81, 126--127, 180--181, 23--22, 49--50, 81--82,
     127--128, 181--182, 24--23, 50--51, 82--83, 129--128, 182--183, 25--24,
```



```
[25] from ts2vg import HorizontalVG



     g = HorizontalVG()
     g.build(meta['metascore'][1:200])

     ig_g = g.as_igraph()

[26] print('Number of Nodes:',ig_g.vcount())
     print('Number of Links:',ig_g.ecount())
     print('Average Degree:',np.mean(ig_g.degree()))
     print('Network Diameter:',ig_g.diameter())
     print('Average Path Length:',ig_g.average_path_length())

     Number of Nodes: 199
     Number of Links: 281
     Average Degree: 2.8241206030150754
     Network Diameter: 45
     Average Path Length: 9.906552966854475
```



```
nx_g = g.as_networkx()
import networkx as nx
nx.draw_kamada_kawai(nx_g)
```

Conclusion :The Visibility Graph and Horigontal Visibility Graph have been printed.The Permutation Entropy, Complexity,number of Nodes,Links, the Average Degree,Network Diameter and the Average Path Length of the 'metascore' have been computed.

##The Average Path Length of the Visibility Graph is lower than that of the Horizontal Visibility Graph which means the average shortest distance between two nodes in the graph is shorter in the Visibility Graph.

Summary: After running the analysis above we have put together the concepts of Probability, Cluster Analysis, Text Analysis and Time Series Analysis we could highlight the trends that the Video Game companies follow to target their audiences. We could analysis and run the descriptive, predictive and prescriptive Analysis to make understand the dataset in depth and made some key highlights:

Sentimental Analysis helped us understand the purchase patterns of customers based on words and their liking in a specific Genre

Observation of the probability of Games Developed on the basis of their Genre and for the various Age Groups

How the Critic scores affected the User's Score

How can we predict the future seasonal trends of sales in the coming 50 days through SMA and for 5 years through Arima model.