How does Broadway show performance affect its financial performance?

a) Problem Statement

According to The Broadway League, Broadway has a total attendance of 14,768,254 and grosses of US$1,829,312,140 in the 2018–2019 season; furthermore, the attendance and gross has been increasing since the year of 2016. The financial statistics exhibit the stable operations and profitability of Broadway and indicate a sustainably growing market of Broadway show performances. Broadway provides public online access to its weekly gross information. This project aims to utilize the essential amount of Broadway weekly transaction data and to identify the critical factors that affect Broadway's financial performance through multivariate data analysis methods. Based on the data scraped from webpages, we conduct an exploratory analysis, Principle Component Analysis, Factor Analysis, and K-Means Clustering to dig into the Broadway show performances.

b) Description of the Data

I scrapped information on Broadway shows from the web pages of www.playbill.com, a monthly U.S. magazine for theatergoers, through web scraping techniques in Python. The Python script for retrieving and processing the information of Broadway shows is attached at the end of this report. The consequential `broadway2019.csv` dataset includes the financial information of any Broadway show that was on show in the year 2019.

The `broadway2019.csv` dataset contains 17 variables and 1,705 observations. It provides the grosses information of 82 shows run over 52 weeks as a total of 12,424 performances in the year 2019. The definition of each variable is listed below:

- ID (Identification number of each observation)
- Show Name
- Week (The date each grosses week ending on)
- Theatre (The theater where a show was on show)
- Gross ($)
- Gross Difference (From week prior)
- Gross Potential (%)
- Average Ticket Price
- Top Ticket Price
- Seats Sold
- Seats in the Theatre
- Number of Performances
- Number of Previews
- Capacity (%)
- Capacity Difference (From week prior)
- Month
- Season

```r
# Import and preview the dataset
broadway2019 = read.csv("broadway2019.csv")

# Number of observations and number of variables
dim(broadway2019)
```

```
## [1] 1705   17
```

```r
# Count of Shows, Weeks, and Performances
intro = c(length(unique(broadway2019$Show.Name)), length(unique(broadway2019$Week)),
sum(broadway2019$Performances))
names(intro) = c("Count of Shows", "Count of Weeks", "Count of Performances")
intro
```

```
##        Count of Shows        Count of Weeks Count of Performances
##                    82                    52                 12424
```

```r
head(broadway2019)
```

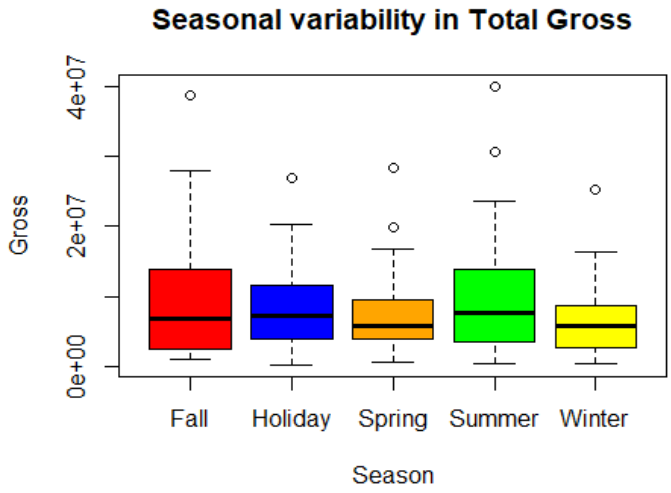
```
##         ID     Week                          Show.Name
## 1 20190001 01/06/19                            Aladdin
## 2 20190002 01/06/19                         American Son
## 3 20190003 01/06/19                         Anastasia
## 4 20190004 01/06/19 Beautiful: The Carole King Musical
##                       Theatre     Gross Potential.Gross Gross.Diff
## 1      New Amsterdam Theatre 1634003.0         1711079  -950546.0
## 2               Booth Theatre  563571.9          888864  -143420.2
## 3         Broadhurst Theatre  785201.1          962806  -554576.1
## 4   Stephen Sondheim Theatre  623691.2         1106908  -395200.0
##   Avg.Ticket.Price Top.Ticket.Price Seats.Sold Seats.in.the.Theatre
## 1           120.01            227.5      13616                 1727
## 2           103.52            248.0       5444                  774
## 3           107.12            273.0       7330                 1143
## 4            86.47            247.0       7213                 1026
##   Performances Previews Capacity Capacity.Diff Month Season
## 1            8        0     0.99         -0.01     1 Winter
## 2            8        0     0.88         -0.07     1 Winter
## 3            7        0     0.92         -0.07     1 Winter
## 4            8        0     0.88         -0.07     1 Winter
```

c) Exploratory Analysis of the Data

```r
# Performances Count of Each Season
showFreq = aggregate(Performances ~ Show.Name + Season, data = broadway2019, sum)
showFreq = spread(showFreq, Season, Performances)
showFreq[is.na(showFreq)] = 0 # Replace NA with 0
(season = colSums(showFreq[, c("Spring", "Summer", "Fall", "Winter", "Holiday")]))
```

```
##  Spring  Summer    Fall  Winter Holiday
##    2069    3466    2695    1839    2355
```

```r
# Gross of Each Season
showGrossSe = aggregate(Gross ~ Show.Name + Season, data = broadway2019, sum)
boxplot(Gross ~ Season, data=showGrossSe, col=c("red","blue","orange","green","yellow
"), main="Seasonal variability in Total Gross")
```




```
showGrossSe = spread(showGrossSe, Season, Gross)
showGrossSe[is.na(showGrossSe)] = 0 # Replace NA with 0
(GrossSe = colSums(showGrossSe[, c("Spring", "Summer", "Fall", "Winter", Holiday")]))

##    Spring    Summer      Fall    Winter   Holiday
## 309771824 451201193 404365405 245697447 346642803

# Which shows were most busy throughout the year of 2019 on Broadway
showFreqWK = aggregate(Performances ~ Show.Name + Week, data = broadway2019, sum)
showFreqWK = na.omit(spread(showFreqWK, Week, Performances))
showFreqWK$Show.Name

##  [1] "Aladdin"
##  [2] "Chicago"
##  [3] "Come From Away"
##  [4] "Dear Evan Hansen"
##  [5] "Frozen"
##  [6] "Hamilton"
##  [7] "Harry Potter and the Cursed Child, Parts One and Two"
##  [8] "Mean Girls"
##  [9] "The Book of Mormon"
## [10] "The Lion King"
## [11] "The Phantom of the Opera"
## [12] "To Kill A Mockingbird"
## [13] "Waitress"
## [14] "Wicked"

# Which shows have the most of grosses in the year of 2019
showGross = aggregate(Gross ~ Show.Name, data = broadway2019, sum)
topGross = top_n(showGross, 10, Gross)
(topGross[order(-topGross$Gross),])

##                                                Show.Name     Gross
## 5                                               Hamilton 159176003
## 8                                             The Lion King 112571006
## 9                                   To Kill A Mockingbird  94444101
## 10                                                Wicked  90673407
## 6   Harry Potter and the Cursed Child, Parts One and Two  78416312
## 2                                                Aladdin  73851144
```

```
## 4                                                    Frozen  63776141
## 1  Ain't Too Proud-The Life and Times of the Temptations  63569789
## 3                                        Dear Evan Hansen  63089977
## 7                                     The Book of Mormon  56210668
```

The exploratory analysis results clearly show that summer and fall are busy seasons for Broadway, as in these two seasons, Broadway has most of their performances on the show and make more money than in other seasons. The boxplot shows that the gross range in summer and fall is bigger than the other seasons, but the median gross does not vary a lot among the five seasons. And there are very few outliers in the five seasons; thus, the weekly gross is usually in a predictable range.

The gross data have revealed that among the eighty-two shows, `Hamilton`, `The Lion King`, and `To Kill A Mockingbird` are the top three shows that have contributed most to the annual gross of Broadway in 2019. The top seven shows that have made the most profits are on show every week throughout the year 2019. Meanwhile, another seven shows were also on the show on Broadway on a weekly basis throughout the year 2019.

Wordcloud

```
for (i in unique(broadway2019$Season)){
  shows_season = subset(broadway2019, Season == i)  # select all the shows in this se
ason
  shows_scount = as.data.frame(aggregate(Performances ~ Show.Name + Season, data = sh
ows_season, sum))
  names(shows_scount) = c("Show_Name", "Season", "Count")

  layout(matrix(c(1, 2), nrow = 2), heights = c(1, 5))
  par(mar = rep(0, 4))
  plot.new()
  text(x = 0.5, y = 0.5, label = paste("Word Cloud:", i), cex=1.5)
  wordcloud(shows_scount$Show_Name, shows_scount$Count, colors=brewer.pal(8,"Spectral
"))
}
```

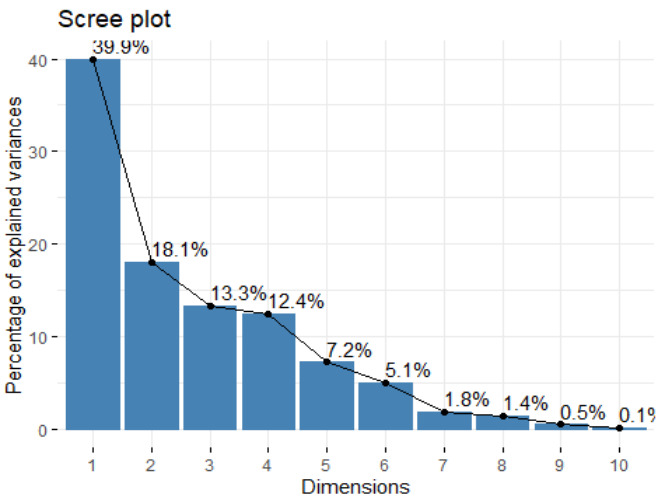
The word clouds by season visualize the popular shows of each season. We can see that shows such as `Hamilton`, `Aladdin`, and `Frozen` are popular throughout the year and regardless of seasons. Those shows are also the busiest shows and most profitable shows of 2019.

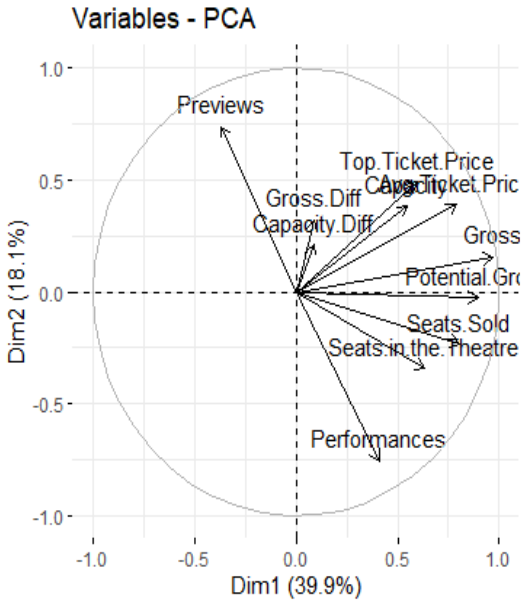
d) Multivariate Analysis of the Data

```
x = as.matrix(broadway2019[,c(5:15)], rownames=broadway2019$ID)
sx = scale(x) # standardizethe data
describe(sx) # Check if mean=0 and sd=1
sx.cor = t(sx) %*% sx /(nrow(sx)-1)
all.equal(sx.cor, cor(x)) # Compare sx.cor with cor(x)
## [1] TRUE
```

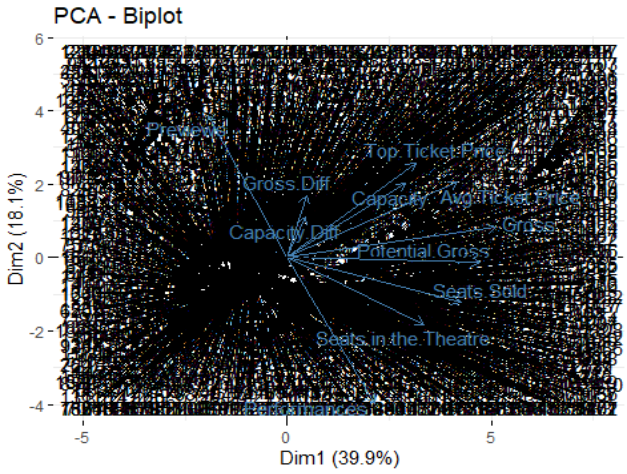
Principal Component Analysis

```
res.pca <- PCA(x, graph = FALSE)
fviz_screeplot(res.pca, addlabels=TRUE)
fviz_pca_var(res.pca)
```





```
fviz_pca_biplot(res.pca, repel = TRUE)
```

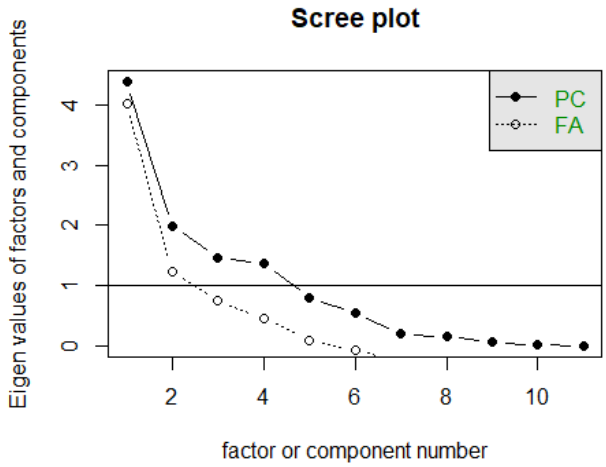


The screen plot represents that the first four components explain more than 85% of the variation of the dataset.

The variable PCA plot is a plot of variable correlation and shows the relationships between all variables. Positively correlated variables are grouped together, while negatively correlated variables are positioned on opposite sides of the plot origin (opposed quadrants). The distance between variables and the origin measures the quality of the variables on the factor map. Variables that are away from the origin are well represented on the factor map. The plot shows that previews are negatively correlated with all the other variables. Seats, number of performances, and gross are positively correlated with each other. The average ticket price and top ticket price are more positively correlated with gross than the number of performances.

Factor Analysis

```
scree(sx.cor)
```




```
(partB.fac.ml = fa(sx, nfactors=4, rotate="varimax", fm="ml", scores=TRUE))
```

```
## Standardized loadings (pattern matrix) based upon correlation matrix
##                       ML4   ML1   ML2   ML3    h2     u2 com
```

```
## Gross                 0.77  0.59  0.12  0.20 0.995 0.0050 2.1
## Potential.Gross       0.62  0.69  0.15 -0.09 0.891 0.1089 2.1
## Gross.Diff            0.10  0.04 -0.08  0.07 0.023 0.9766 3.2
## Avg.Ticket.Price      0.96  0.13  0.08  0.15 0.959 0.0405 1.1
## Top.Ticket.Price      0.82 -0.02 -0.01  0.06 0.683 0.3171 1.0
## Seats.Sold            0.15  0.93  0.16  0.30 0.995 0.0047 1.3
## Seats.in.the.Theatre  0.01  0.94  0.03 -0.04 0.892 0.1083 1.0
## Performances          0.04  0.18  0.98  0.00 0.995 0.0050 1.1
## Previews             -0.07 -0.08 -0.93 -0.01 0.873 0.1267 1.0
## Capacity              0.39  0.13 -0.02  0.91 0.995 0.0050 1.4
## Capacity.Diff        -0.01  0.01  0.00  0.27 0.073 0.9269 1.0
##
##                        ML4  ML1  ML2  ML3
## SS loadings           2.76 2.65 1.90 1.07
## Proportion Var        0.25 0.24 0.17 0.10
## Cumulative Var        0.25 0.49 0.66 0.76
## Proportion Explained  0.33 0.32 0.23 0.13
## Cumulative Proportion 0.33 0.65 0.87 1.00
##
## Mean item complexity =  1.5
## Test of the hypothesis that 4 factors are sufficient.
##
## The degrees of freedom for the null model are  55  and the objective function was  14.41 wi
th Chi Square of  24492.41
## The degrees of freedom for the model are 17  and the objective function was  2.47
##
## The root mean square of the residuals (RMSR) is  0.05
## The df corrected root mean square of the residuals is  0.09
##
## The harmonic number of observations is  1705 with the empirical chi square  495.61  with pr
ob <  1.6e-94
## The total number of observations was  1705  with Likelihood Chi Square =  4188.17  with pro
b <  0
##
## Tucker Lewis Index of factoring reliability =  0.447
## RMSEA index =  0.379  and the 90 % confidence intervals are  0.37 0.389
## BIC =  4061.67
## Fit based upon off diagonal values = 0.98
## Measures of factor score adequacy
##                                                 ML4  ML1  ML2  ML3
## Correlation of (regression) scores with factors   0.99 1.00 1.00 1.00
## Multiple R square of scores with factors          0.99 0.99 0.99 0.99
## Minimum correlation of possible factor scores     0.98 0.99 0.99 0.98
```

According to the scree plot, the first four factors account for most of the total variability in data. The remaining factors account for a very small proportion of the variability and are unlikely to be important. Therefore, I used four factors in the factor analysis. The factor analysis results show that the gross, potential gross, and ticket prices are loaded heavily on the fourth factor, while seats load heavily on the first factor, and the number of performances loads heavily on the second factor. So, the first factor should be a factor related to the number of seats, the second factor is a performance factor, the third factor is capacity, and the fourth factor is a related price factor. The results also attach the importance of seat arrangement and ticket pricing on the improvement of gross.

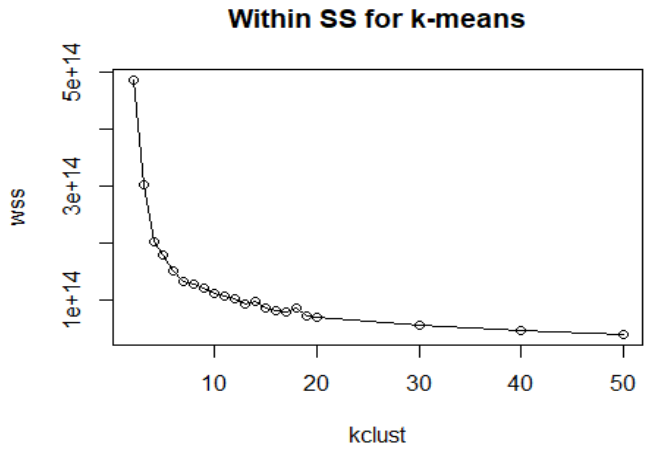
K-means Clustering

```
# compute multiple cluster solutions
kclust=c(2:20,30,40,50) # create a vector of k values to try
nclust=length(kclust) # number of kmeans solutions to compute
bss=wss=rep(0,nclust) # initialize vectors bss and wss to zeroes
```

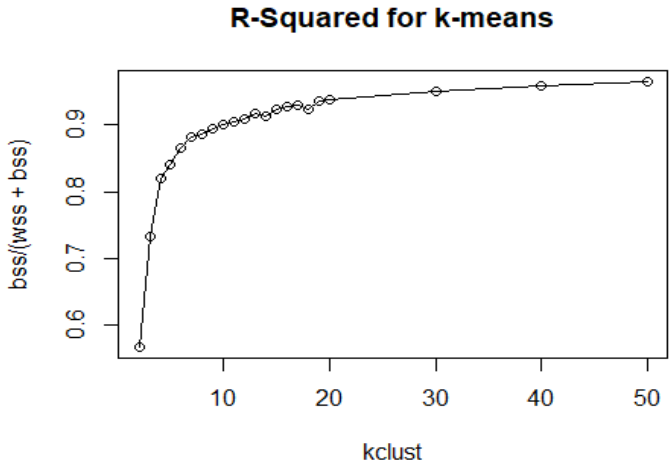
```r
set.seed(34612) # set the seed so we can repeat the results
grpQ=as.list(rep(NULL,nclust)) # create empty list to save results

# compute SS for each cluster
for (i in 1:nclust) {
grpQ[[i]]=kmeans(x,kclust[i],nstart=1) # compute kmeans solution
wss[i]=grpQ[[i]]$tot.withinss # save the within SS
bss[i]=grpQ[[i]]$betweenss # save the between SS
}

# plot the results and look for the "Hockey-Stick" effect
par(mfrow=c(1,1))
plot(kclust,wss,type="l",main="Within SS for k-means")
points(kclust,wss)
```




```r
plot(kclust,bss/(wss+bss),type="l",main="R-Squared for k-means") # R-Squared is ratio of explained variation
points(kclust,bss/(wss+bss))
```




```r
# compute a k=18 solution
set.seed(569123)   # initialize the random number generator so we all get the same results
grpB=kmeans(x, centers=14, nstart=20)
```

```
# summarize the centroids
grpBcenter=t(grpB$centers)

# compare the cluster solutions with shows
table(broadway2019$Show.Name, grpB$cluster)
```

```
##
##                                                  1  2  3  4  5  6  7  8
##   A Christmas Carol                              0  0  0  0  0  2  0  5
##   A Soldier's Play                               0  0  0  0  0  0  0  0
…
```

```
topGross$Show.Name
```

```
##  [1] "Ain't Too Proud-The Life and Times of the Temptations"
##  [2] "Aladdin"
##  [3] "Dear Evan Hansen"
##  [4] "Frozen"
##  [5] "Hamilton"
##  [6] "Harry Potter and the Cursed Child, Parts One and Two"
##  [7] "The Book of Mormon"
##  [8] "The Lion King"
##  [9] "To Kill A Mockingbird"
## [10] "Wicked"
```

```
# Compare the top 10 gross shows
top10data = subset(broadway2019, Show.Name %in% topGross$Show.Name)
join(join(join(join(aggregate(Avg.Ticket.Price ~ Show.Name, top10data, median),
     aggregate(Top.Ticket.Price ~ Show.Name, top10data, median)),
     aggregate(Performances ~ Show.Name, top10data, sum)),
      aggregate(Capacity ~ Show.Name, top10data, sum)),
       aggregate(Seats.in.the.Theatre ~ Show.Name, top10data, mean))
```

```
##                                              Show.Name Avg.Ticket.Price
## 1  Ain't Too Proud-The Life and Times of the Temptations          133.695
## 2                                             Aladdin          104.265
## 3                                       Dear Evan Hansen          153.145
## 4                                                Frozen           96.845
## 5                                              Hamilton          287.270
## 6   Harry Potter and the Cursed Child, Parts One and Two          115.715
## 7                                       The Book of Mormon          123.495
## 8                                           The Lion King          161.000
## 9                                    To Kill A Mockingbird          160.645
## 10                                                Wicked          120.790
##    Top.Ticket.Price Performances Capacity Seats.in.the.Theatre
## 1             297.0          323    43.10             1424.000
## 2             227.5          418    50.34             1727.000
## 3             348.0          415    52.45              984.000
## 4             252.5          417    48.78             1684.000
## 5             849.0          415    52.99             1321.923
## 6             296.5          413    52.00             1622.000
## 7             477.5          414    53.03             1047.000
## 8             225.0          419    50.71             1696.000
```

```
## 9             497.0          415    52.75             1435.000
## 10            240.0          418    49.84             1847.981
```

In the plot of Within SS for k−means, the within-cluster sum of squares jumps up when the k increases from 14 to 15, indicating that the variability of the observations within each cluster increases a lot between the k value of 14 and 15. Thus, 14 will be a good value of k. Furthermore, in the plot of R−Squared for k−means, the R-Squared jumps down when the k increases from 14 to 15, indicating that explained variation of the model decreases a lot between the k value of 14 and 15, which also means that 14 will be a good value of k.

Compare the cluster solutions with shows, we can found that the top ten shows that have earned most of the gross in 2019 concentrate in the cluster 1, 3, 4, and 6. These shows have more familiar characters than the other performances. For example, the median average ticket prices, numbers of performances, the capacity of the top ten shows are very close to each other, the median top ticket prices and mean number of seats in the theater vary a little bit more than other attributes

e) Conclusion

In the exploratory analysis, we illustrated that the seasonal variation exists in the weekly gross of Broadway shows while the variation is not significantly high. Summer and fall are the busiest and most fruitful seasons. Since warm weather would encourage people to travel and hang out more often than the cold seasons, weather can be an important reason why a significant portion of gross was created in the summer and fall. The exploratory analysis also shows that many of the Broadway shows, which contribute most to the gross, are performed regularly and frequently. In the exploratory analysis, we identified the weather factor and performance count are essential factors to gross.

In the multivariate analysis, we noticed the positive correlation between gross and ticket price and between capacity and gross via PCA analysis. Price and capacity positively affect gross on both dimensions of the first two principal components. The results suggest that Broadway should carefully price their tickets and arrange the seats in the theaters if they want to improve the gross. A good pricing strategy and appropriate seating arrangement would boost sales better than just a high frequency of performances.

In the factor analysis, we identified four essential factors. The first factor should be related to the number of seats, the second factor is a performance factor, the third factor is capacity, and the fourth factor is a related price factor. The results also attach the importance of seat arrangement and ticket pricing to the improvement of gross and support the previous results we found.

In the clustering analysis, we found that the shows that contribute most to the gross have more familiar characters than the other performances. They are very similar to each other regarding the median average ticket prices, numbers of performances, and the capacity; nevertheless, these shows are a little bit dissimilar to each other on the median top ticket prices and mean the number of seats in the theater. Overall, the results reveal that the seat arrangement and ticket pricing are of great importance to gross improvement.

Appendix

Python Script for Web Scraping Broadway Shows

```
import os
import requests
import urllib
import math
import copy
import pandas as pd
import numpy as np
import datetime
from bs4 import BeautifulSoup


#########################################################################
###################### Prepare Functions ################################
#########################################################################

class html_tables(object):

    def __init__(self, url):

        self.url      = url
        self.r        = requests.get(self.url)
        self.url_soup = BeautifulSoup(self.r.text)

    def read(self):

        self.tables      = []
        self.tables_html = self.url_soup.find_all("table")

        # Parse each table
        for n in range(0, len(self.tables_html)):

            n_cols = 0
            n_rows = 0

            for row in self.tables_html[n].find_all("tr"):
                col_tags = row.find_all(["td", "th"])
                if len(col_tags) > 0:
                    n_rows += 1
                    if len(col_tags) > n_cols:
                        n_cols = len(col_tags)

            # Create dataframe
            df = pd.DataFrame(index = range(0, n_rows), columns = range(0, n_cols))

            # Create list to store rowspan values
            skip_index = [0 for i in range(0, n_cols)]

            # Start by iterating over each row in this table...
            row_counter = 0
            for row in self.tables_html[n].find_all("tr"):
```

```
            # Skip row if it's blank
            if len(row.find_all(["td", "th"])) == 0:
                next

            else:

                # Get all cells containing data in this row
                columns = row.find_all(["td", "th"])
                col_dim = []
                row_dim = []
                col_dim_counter = -1
                row_dim_counter = -1
                col_counter = -1
                this_skip_index = copy.deepcopy(skip_index)

                for col in columns:

                    # Determine cell dimensions
                    colspan = col.get("colspan")
                    if colspan is None:
                        col_dim.append(1)
                    else:
                        col_dim.append(int(colspan))
                    col_dim_counter += 1

                    rowspan = col.get("rowspan")
                    if rowspan is None:
                        row_dim.append(1)
                    else:
                        row_dim.append(int(rowspan))
                    row_dim_counter += 1

                    # Adjust column counter
                    if col_counter == -1:
                        col_counter = 0
                    else:
                        col_counter = col_counter + col_dim[col_dim_counter - 1]

                    while skip_index[col_counter] > 0:
                        col_counter += 1

                    # Get cell contents
                    cell_data = col.get_text()

                    # Insert data into cell
                    df.iat[row_counter, col_counter] = cell_data

                    # Record column skipping index
                    if row_dim[row_dim_counter] > 1:
                        this_skip_index[col_counter] = row_dim[row_dim_counter]

            # Adjust row counter
            row_counter += 1

            # Adjust column skipping index
```

```
                skip_index = [i - 1 if i > 0 else i for i in this_skip_index]

            # Append dataframe to list of tables
            self.tables.append(df)

        return(self.tables)



##########################################################################
################## Scrap Information from Webpages ######################
##########################################################################

# Select all the Sundays of 2019
def allsundays(year):
    return pd.date_range(start=str(year), end=str(year+1),
                         freq='W-SUN').strftime('%Y-%m-%d').tolist()
sunday2019 = list(allsundays(2019))

# Scrap the information of all broadway shows in 2019
data = pd.DataFrame()
for week in sunday2019:
    url = "https://www.playbill.com/grosses?week="+week
#    request = requests.get(url)
#    if request.status_code == 200:     # Check if the url exists
    df = html_tables(url).read()[0]
    df['Week'] = week
    data = data.append(df)

data.to_csv("broadwaydata2019.csv", index=False)
data.isnull().values.any() # None null




##########################################################################
################ Processing Web Scraped Information #####################
##########################################################################

listData = data.iloc[:, 0:len(data.columns)-1]

for i in range(len(listData.columns)):
    listData.iloc[:,i] = listData.iloc[:,i].str.split('\n')

Broadway2019 = pd.concat([
    listData.iloc[:,0].apply(pd.Series).iloc[:,[1,2]],
    listData.iloc[:,1].apply(pd.Series).iloc[:,[1,2]],
    listData.iloc[:,2].apply(pd.Series).iloc[:,1],
    listData.iloc[:,3].apply(pd.Series).iloc[:,[1,2]],
    listData.iloc[:,4].apply(pd.Series).iloc[:,[1,2]],
    listData.iloc[:,5].apply(pd.Series).iloc[:,[1,2]],
    listData.iloc[:,6].apply(pd.Series).iloc[:,1],
    listData.iloc[:,7].apply(pd.Series).iloc[:,1]],
    axis=1, ignore_index=True)
```

```
Broadway2019.columns = ["Show Name", "Theatre", "Gross", "Potential Gross", "Gross Di
ff",
                        "Avg Ticket Price", "Top Ticket Price", "Seats Sold", "Seats
in the Theatre",
                        "Performances", "Previews", "Capacity", "Capacity Diff"]

Broadway2019['Week'] = data["Week"]
Broadway2019['Month'] = pd.DatetimeIndex(Broadway2019['Week']).month
Broadway2019.loc[(Broadway2019['Month']==1) | (Broadway2019['Month']==2), 'Season'] =
'Winter'
Broadway2019.loc[(Broadway2019['Month']==3) | (Broadway2019['Month']==4), 'Season'] =
'Spring'
Broadway2019.loc[(Broadway2019['Month']==5) | (Broadway2019['Month']==6) | (Broadway2
019['Month']==7), 'Season'] = 'Summer'
Broadway2019.loc[(Broadway2019['Month']==8) | (Broadway2019['Month']==9) | (Broadway2
019['Month']==10), 'Season'] = 'Fall'
Broadway2019.loc[(Broadway2019['Month']==11) | (Broadway2019['Month']==12), 'Season']
= 'Holiday'

Broadway2019.isnull().values.any() # None null

Broadway2019.to_csv("broadway2019.csv", index=False)
```