Principal Component Analysis: Market Shift Detection R Code Explanation

Carlos Monsivais
Belami E- Commerce
VAM Department

September 5, 2018

Table of Contents

How Principal Component Analysis Works

Principal Component Analysis (also known as PCA) essentially reduces the dimensionality of your data set. For example, if you have an Excel file each column is one dimension of data. If we have a data set with 20 Excel columns, we would in theory have a 20^{th} dimensional data set. As a result, anything that is over 3 dimensions is very difficult to imagine since as human beings we live in a 3-dimensional world. We need some method that will reduce the dimensionality of this data set into something that we can comprehend and visualize to perform statistical procedures on it. Before the analysis of PCA, the data must be scaled so that it can create an orthogonal set of numerical values, they must all have the same mean and be scaled within a certain distance.

As a result, PCA is a statistical procedure that converts a set of variables that are possibly correlated and converts them into a set of variables that are linearly uncorrelated. After the conversion into linearly uncorrelated variables, these new values are called principal components. The principal components are converted so that the first principal component known as PC1 has the largest variance which means that it accounts for as much of the variability in the data as possible. Afterwards, PC2 has the second highest variance which is interpreted as the second highest principal component that accounts for the second highest variability in the data.

To perform PCA, we do the following:

1. Gather the n samples of m dimensional data $\overrightarrow{x_1}, \dots \dots . . \overrightarrow{x_n}$ in your data set $|R^m$ where we compute the following:
   - Sample Average: $\vec{u} = \frac{1}{n}(\overrightarrow{x_1} + \cdots \dots . + \overrightarrow{x_n})$
   - Build the Matrix B: $B = [\overrightarrow{|x_1 - u|}, \dots \dots \dots . , \overrightarrow{|x_n - \vec{u}|}]$
   - Compute matrix S (Covariance Matrix): $S = \frac{1}{n-1} \times B \times B^T$

2. Find the eigenvalues $\lambda_1, \dots \dots \dots . , \lambda_m$ of S arranged in decreasing order as well as an orthogonal set of eigenvectors $\vec{u}_1, \dots \dots \dots . , \vec{u}_m$

3. Interpret the results as following:
   - Are a small number of the λ_i much bigger than the others?
   - If so, this indicates a dimension reduction is possible.
   - Which of then variables are most important in the first, second, third, etc... principal components?
   - Which factors appear with the same or opposite signs as the others?

Advantages of PCA	Disadvantages of PCA
Useful for dimension reduction for high dimensional data analysis	Only numerical values can be read in
Helps reduce the number of predictor items using principal components	Prediction models are usually less interpretable
Helps to make predictor items independent to avoid multicollinearity problems	
Allows you to interpret many variables in a 2- dimensional plot	
Can be used to develop predictive models	

Step 1: Reading in the Data

```
library(readxl)
```

We need this library to use the read_excel function so that we can read in the data from the excel file.

```
url <- "file:///C:/Users/carlos.monsivais/Desktop/PCA Galtech.xlsx"
destfile <- "PCA_Galtech.xlsx"
curl::curl_download(url, destfile)
PCA_Galtech <- read_excel(destfile)
```

Here we are reading in the excel file with all the data on it.

```
data = PCA_Galtech
```

Here I am just renaming the file where all the data is contained from PCA_Galtech to just data for simplicity reasons to not have such a long name.

```
str(data)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':     232 obs. of  9 variables:
 $ purchasedate: POSIXct, format: "2017-04-01" ...
 $ Sum of Sales: num  1345 6475 7366 12304 4211 ...
 $ Sum of Cost : num  784 3521 3828 7015 2147 ...
 $ Clicks      : num  65 78 71 59 38 49 42 57 68 91 ...
 $ Impressions : num  3896 4243 3611 3241 2887 ...
 $ Cost        : num  276 361 297 286 177 ...
 $ Conversions : num  5 4 4 2 3 1 1 3 2 2 ...
 $ Year        : num  2017 2017 2017 2017 2017 ...
 $ Month       : chr  "April" "April" "April" "April" ...
```

Here I am looking at the structure of the data in terms of what each variable is formatted at, whether it’s a numerical value, a factor variable or a character variable.

```
summary(data)
  purchasedate                  Sum of Sales    Sum of Cost        Clicks
 Min.   :2017-04-01 00:00:00   Min.   :    0   Min.   :-793.6   Min.   : 30.00
 1st Qu.:2017-05-28 18:00:00   1st Qu.: 2630   1st Qu.:1415.3   1st Qu.: 65.00
 Median :2017-11-27 00:00:00   Median : 4126   Median :2201.3   Median : 80.00
 Mean   :2017-11-27 00:00:00   Mean   : 4576   Mean   :2520.9   Mean   : 80.67
 3rd Qu.:2018-05-28 06:00:00   3rd Qu.: 5831   3rd Qu.:3277.9   3rd Qu.: 94.00
 Max.   :2018-07-25 00:00:00   Max.   :16141   Max.   :9214.7   Max.   :150.00
  Impressions       Cost          Conversions         Year         Month
 Min.   : 2400   Min.   :146.4   Min.   : 0.000   Min.   :2017   Length:232
 1st Qu.: 3859   1st Qu.:245.6   1st Qu.: 2.000   1st Qu.:2017   Class :character
 Median : 4692   Median :301.2   Median : 4.000   Median :2018   Mode  :character
 Mean   : 4996   Mean   :328.7   Mean   : 4.315   Mean   :2018
 3rd Qu.: 5864   3rd Qu.:377.2   3rd Qu.: 6.000   3rd Qu.:2018
 Max.   :10530   Max.   :893.1   Max.   :16.000   Max.   :2018
```

Here by using the summary function we can get summary statistics on each variable no matter in what format it is.

```
data$Year = as.factor(data$Year)
```

Here we are converting the variable Year into a factor variable which means we are turning this variable into a categorical variable. For example, in the structure of the data above we can see that the variable Year is being read in as a numerical variable however we are now turning it into a factor type.

```
data$Month = as.factor(data$Month)
```

Here we are converting the variable Month into a factor variable which means we are turning this variable into a categorical variable. For example, in the structure of the data above we can see that the variable Month is being read in as a character variable however we are now turning it into a factor type.

```
str(data)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':     232 obs. of  9 variables:
 $ purchasedate: POSIXct, format: "2017-04-01" ...
 $ Sum of Sales: num  1345 6475 7366 12304 4211 ...
 $ Sum of Cost : num  784 3521 3828 7015 2147 ...
 $ Clicks      : num  65 78 71 59 38 49 42 57 68 91 ...
 $ Impressions : num  3896 4243 3611 3241 2887 ...
 $ Cost        : num  276 361 297 286 177 ...
 $ Conversions : num  5 4 4 2 3 1 1 3 2 2 ...
 $ Year        : Factor w/ 2 levels "2017","2018": 1 1 1 1 1 1 1 1 1 1 ...
 $ Month       : Factor w/ 4 levels "April","July",..: 1 1 1 1 1 1 1 1 1 1 ...
```

We can now see that the variables we turned into factor type meaning that they will now be read in by R as categorical variables have changed.

Step 2: Separating the Data by Years 2017 and 2018

```
data2017 = data.frame(data[1:116,])
```

Here I am manually sub setting the data in the sense of separating the data set by Years. Here I am creating a subset of data consisting of only the 2017 data. I do this to individually apply the Principal Component Analysis method and see the main characteristics of each year.

```
complete.cases(data2017)
  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [10] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [19] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [28] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [37] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [55] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [64] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [73] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [82] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [91] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[100] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[109] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

I use the complete.cases function on this sub set data set to see in which row there is any type of inconsistency on unfilled data rows. I want to see where there are any unfilled data rows to check if maybe there was a mistake when the data was inputted or if there wasn’t any data in that column for that row.

```
data2017 = data2017[complete.cases(data2017),]
```

What this line of code does is it tells this subset of 2017 data that we only want to keep the data rows that are complete. In how we are removing any rows of data that are missing values. This is important for Principal Component Analysis , to no have any numerical values missing because it uses them for the calculations of the algorithm.

```
data2018 = data.frame(data[117:232,])
```

Here I am manually sub setting the data in the sense of separating the data set by Years. Here I am creating a subset of data consisting of only the 2018 data. I do this to individually apply the Principal Component Analysis method and see the main characteristics of each year.

```
complete.cases(data2018)
  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [10] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [19] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [28] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [37] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [55] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [64] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [73] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [82] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
 [91] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[100] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[109] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

I use the complete.cases function on this sub set data set to see in which row there is any type of inconsistency on unfilled data rows. I want to see where there are any unfilled data rows to check if maybe there was a mistake when the data was inputted or if there wasn’t any data in that column for that row.

```
data2018 = data2018[complete.cases(data2018),]
```

What this line of code does is it tells this subset of 2018 data that we only want to keep the data rows that are complete. In how we are removing any rows of data that are missing values. This is important for Principal Component Analysis , to no have any numerical values missing because it uses them for the calculations of the algorithm.

Step 3: Correlation Coefficients

```
data2017 = data2017[, -c(1,3,4,8,9)]
```

Here I am removing the following variables because they were either functions of one another when being calculated or they were just very highly correlated with correlation values of up to 0.99. I removed these variables from the analysis.

```
cor_data2017 = round(cor(data2017), 2)
```

Here I am calculating the Pearson Correlation value for the 4 remaining variables. I am also using the round function to round the correlation values to the hundredths place.

```
top2017 = cor_data2017
top2017
             Sum.of.Sales Impressions Cost Conversions
Sum.of.Sales         1.00       -0.05 0.12        0.24
Impressions         -0.05        1.00 0.52        0.28
Cost                 0.12        0.52 1.00        0.44
Conversions          0.24        0.28 0.44        1.00
```

Here I am renaming the correlation matrix we created to top2017 so it's easier to see the upper triangle that we are going to fix below.

```
top2017[upper.tri(cor_data2017)] = ""
```

Here I am assigning a blank space to the upper part of the correlation matrix because it is repetitive. For example, as we can see above the values repeat therefore we are using this part to remove those repetitive values.

```
top2017 = as.data.frame(top2017)
```

Here we are converting the correlations matrix into a data frame to have the values nicely organized in the manner that a data frame would.

```
top2017
             Sum.of.Sales Impressions Cost Conversions
Sum.of.Sales            1                                 
Impressions         -0.05           1                     
Cost                 0.12        0.52    1                
Conversions          0.24        0.28 0.44           1
```

Here is what the correlation matrix looks like with the upper triangle having blanks since they are repetitive anyways.

```
data2018 = data2018[, -c(1,3,4,8,9)]
```

Here I am removing the following variables because they were either functions of one another when being calculated or they were just very highly correlated with correlation values of up to 0.99. I removed these variables from the analysis.

```
cor_data2018 = round(cor(data2018), 2)
```

Here I am calculating the Pearson Correlation value for the 4 remaining variables. I am also using the round function to round the correlation values to the hundredths place.

```
top2018 = cor_data2018
top2018
             Sum.of.Sales Impressions Cost Conversions
Sum.of.Sales         1.00        0.18 0.32        0.48
Impressions          0.18        1.00 0.58        0.30
Cost                 0.32        0.58 1.00        0.50
Conversions          0.48        0.30 0.50        1.00
```

Here I am renaming the correlation matrix we created to top2018 so it's easier to see the upper triangle that we are going to fix below.

```
top2018[upper.tri(cor_data2018)] = ""
```

Here I am assigning a blank space to the upper part of the correlation matrix because it is repetitive. For example, as we can see above the values repeat therefore we are using this part to remove those repetitive values.

```
top2018 = as.data.frame(top2018)
```

Here we are converting the correlations matrix into a data frame to have the values nicely organized in the manner that a data frame would.

```
top2018
              Sum.of.Sales Impressions Cost Conversions
Sum.of.Sales             1                              
Impressions           0.18           1                  
Cost                  0.32        0.58    1             
Conversions           0.48         0.3  0.5           1
```

Here is what the correlation matrix looks like with the upper triangle having blanks since they are repetitive anyways.

Step 4: Principal Component Analysis

```
principal_component2017 = prcomp(data2017, center = TRUE, scale.=TRUE)
```

We are using the prcomp function to actually apply the Principal Component Analysis algorithm. We are applying it to the 2017 data to see how this year performed individually. This is the meaning of the following variables inside the function:

- center : Here we want the center to be TRUE which means that we want our variables to be shifted so that they are centered around 0. This is just easier to visualize since 0 is an absolute value.
- scale. : We are making this equivalent to TRUE because we want to scale all our data so that we are comparing and reading in the data points evenly.

```
important.PC2017 = data.frame(principal_component2017$sdev^2)
```

By taking the square of the standard deviations of the Principal Components we can obtain the eigenvalue of each Principal Component. We will be following Kaiser's Rule which is a rule of thumb stating that any we should take into consideration and eigenvalue that is greater than 1. By calculating the eigenvalues and putting them into a data frame we can pick out which Principal Components to look at.

```
rownames(important.PC2017) = c("PC1", "PC2", "PC3", "PC4")
```

These will be the row names for the data frame I am creating that will display the eigenvalues of each Principal Component.

```
colnames(important.PC2017) = c("Eigenvalues")
```

This will be the name of the column in the data frame I am trying to create that will display the eigenvalue of each Principal Component.

```
important.PC2017
    Eigenvalues
PC1   1.8674404
PC2   1.0785001
PC3   0.6176599
PC4   0.4363996
```

Here is the data frame I created showing the eigenvalues of each of the Principal Components. Using Kaiser's Rule, we should look at PC1 and PC2 since they are above 1.

```
print(principal_component2017)
Standard deviations (1, .., p=4):
[1] 1.3665432 1.0385086 0.7859134 0.6606055

Rotation (n x k) = (4 x 4):
                   PC1        PC2        PC3        PC4
Sum.of.Sales 0.2061700  0.8476706 -0.4778788  0.1028609
Impressions  0.5306945 -0.4437460 -0.4339301  0.5771979
Cost         0.6199796 -0.1289767 -0.1256815 -0.7636717
Conversions  0.5398912  0.2605937  0.7533530  0.2703105
```

From the print command on the Principal Component we can see the standard deviations of each Principal Component. Remember we can square these to see if they pass Kaiser's Rule or not. We can also see the rotation value, otherwise called the loadings which tell us the correlation between each variable and that Principal Component.

```
summary(principal_component2017)
Importance of components:
                          PC1    PC2    PC3    PC4
Standard deviation     1.3665 1.0385 0.7859 0.6606
Proportion of Variance 0.4669 0.2696 0.1544 0.1091
Cumulative Proportion  0.4669 0.7365 0.8909 1.0000
```

By using the summary function, we can see how much of the variation each Principal Component explains. For example, we can see how PC1 and PC2 will always explain most of the variation in the data.

```
varimax2017 = varimax(principal_component2017$rotation[,1:2])
varimax2017
$`loadings`

Loadings:
              PC1    PC2   
Sum.of.Sales -0.151  0.859
Impressions   0.664 -0.194
Cost          0.620  0.131
Conversions   0.390  0.455

                PC1  PC2
SS loadings    1.00 1.00
Proportion Var 0.25 0.25
Cumulative Var 0.25 0.50

$rotmat
           [,1]      [,2]
[1,]  0.9160187 0.4011356
[2,] -0.4011356 0.9160187
```

By using the varimax function, we are rotating on the rotations outputted by the prcomp function. What this essentially does is that it tells us which variables are the most important in each component. I am only using the varimax function on PC1 and PC2 because those components are the one's that explain most of the variability in the data therefore I want to see which variables are important to these two components. We can also see in the output the sum of square loadings. The most important variable is the variable with the highest absolute loadings value in each component.

```
principal_component2018 = prcomp(data2018, center = TRUE, scale.=TRUE)
```

We are using the prcomp function to actually apply the Principal Component Analysis algorithm. We are applying it to the 2018 data to see how this year performed individually. This is the meaning of the following variables inside the function:

- center : Here we want the center to be TRUE which means that we want our variables to be shifted so that they are centered around 0. This is just easier to visualize since 0 is an absolute value.
- scale. : We are making this equivalent to TRUE because we want to scale all our data so that we are comparing and reading in the data points evenly.

```
important.PC2018 = data.frame(principal_component2018$sdev^2)
```

By taking the square of the standard deviations of the Principal Components we can obtain the eigenvalue of each Principal Component. We will be following Kaiser's Rule which is a rule of thumb stating that any we should take into consideration and eigenvalue that is greater than 1. By calculating the eigenvalues and putting them into a data frame we can pick out which Principal Components to look at.

```
rownames(important.PC2018) = c("PC1", "PC2", "PC3", "PC4")
```

These will be the row names for the data frame I am creating that will display the eigenvalues of each Principal Component.

```
colnames(important.PC2018) = c("Eigenvalues")
```

This will be the name of the column in the data frame I am trying to create that will display the eigenvalue of each Principal Component.

```
important.PC2018
    Eigenvalues
PC1   2.1884039
PC2   0.9320306
PC3   0.5112613
PC4   0.3683042
```

Here is the data frame I created showing the eigenvalues of each of the Principal Components. Using Kaiser's Rule, we should look at PC1. However, PC2 is so close to 1 that I will include it because I want to be able to compare apples to apples. Because in the 2017 data I am looking at PC1 and PC2 and for 2018 I want to look at the same Principal Components and since it's so close to1 I will include this PC2. Remember, Kaiser's Rule is a rule of thumb therefore the guidelines are not extremely strict.

```
print(principal_component2018)
Standard deviations (1, .., p=4):
[1] 1.4793255 0.9654173 0.7150254 0.6068807

Rotation (n x k) = (4 x 4):
                  PC1        PC2        PC3        PC4
Sum.of.Sales 0.4324174  0.6462527  0.6254817  0.0643843
Impressions  0.4685220 -0.6148829  0.3648154 -0.5189565
Cost         0.5632444 -0.3000682 -0.1569416  0.7537135
Conversions  0.5256025  0.3379875 -0.6716038 -0.3980637
```

From the print command on the Principal Component we can see the standard deviations of each Principal Component. Remember we can square these to see if they pass Kaiser's Rule or not. We can also see the rotation value, otherwise called the loadings which tell us the correlation between each variable and that Principal Component.

```
summary(principal_component2018)
Importance of components:
                          PC1    PC2    PC3     PC4
Standard deviation     1.4793 0.9654 0.7150 0.60688
Proportion of Variance 0.5471 0.2330 0.1278 0.09208
Cumulative Proportion  0.5471 0.7801 0.9079 1.00000
```

By using the summary function, we can see how much of the variation each Principal Component explains. For example, we can see how PC1 and PC2 will always explain most of the variation in the data.

```
varimax2018 = varimax(principal_component2018$rotation[,1:2])
varimax2018
$`loadings`

Loadings:
             PC1    PC2   
Sum.of.Sales -0.124  0.768
Impressions   0.762 -0.131
Cost          0.617  0.164
Conversions   0.154  0.606

                PC1  PC2
SS loadings    1.00 1.00
Proportion Var 0.25 0.25
Cumulative Var 0.25 0.50

$rotmat
           [,1]      [,2]
[1,]  0.7319580 0.6813497
[2,] -0.6813497 0.7319580
```

By using the varimax function, we are rotating on the rotations outputted by the prcomp function. What this essentially does is that it tells us which variables are the most important in each component. I am only using the varimax function on PC1 and PC2 because those components are the one's that explain most of the variability in the data therefore I want to see which variables are important to these two components. We can also see in the output the sum of square loadings. The most important variable is the variable with the highest absolute loadings value in each component.

Step 5: Principal Component Analysis Effects on Correlation

```
cor_training2017 = round(cor(principal_component2017$x),2)
```

I am again getting the correlation values however this time I am doing so using the transformed data that has been passed through the Principal Component algorithm. Therefore, the values being used to calculate this Pearson Correlation are transformed values because of passing them through the Principal Component algorithm. This is for the 2017 data set.

```
top2017 = cor_training2017
top2017[upper.tri(cor_training2017)] = ""
top2017 = as.data.frame(top2017)
top2017
    PC1 PC2 PC3 PC4
PC1   1
PC2   0   1
PC3   0   0   1
PC4   0   0   0   1
```

Using the same code as in Step 3, the following correlation matrix should not be surprising because what PCA does is that it decorrelates the values by using an orthogonal transformation and therefore gives us a correlation matrix like the one above.

```
cor_training2018 = round(cor(principal_component2018$x),2)
```

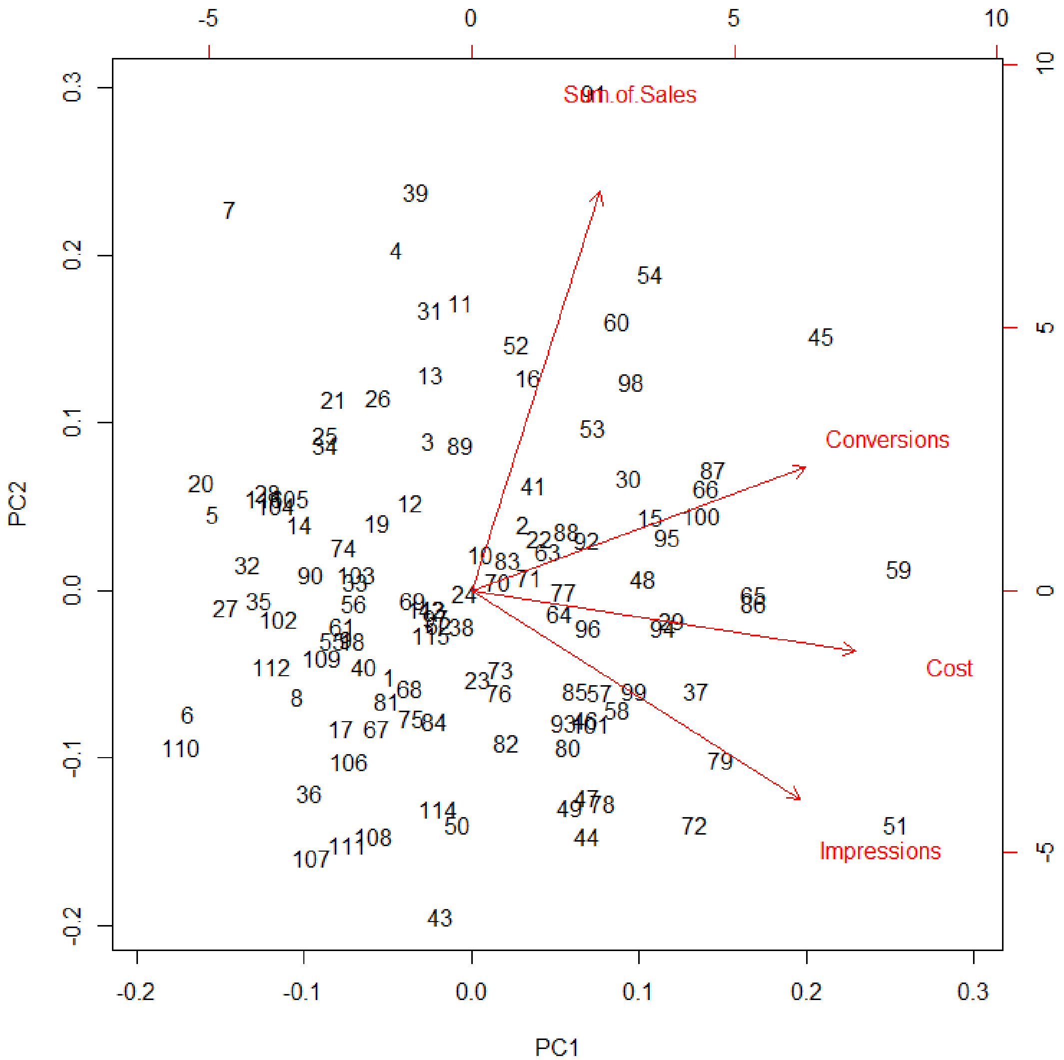
I am again getting the correlation values however this time I am doing so using the transformed data that has been passed through the Principal Component algorithm. Therefore, the values being used to calculate this Pearson Correlation are transformed values because of passing them through the Principal Component algorithm. This is for the 2018 data set.

```
top2018 = cor_training2018
top2018[upper.tri(cor_training2018)] = ""
top2018 = as.data.frame(top2018)
top2018
    PC1 PC2 PC3 PC4
PC1   1
PC2   0   1
PC3   0   0   1
PC4   0   0   0   1
```

Using the same code as in Step 3, the following correlation matrix should not be surprising because what PCA does is that it decorrelates the values by using an orthogonal transformation and therefore gives us a correlation matrix like the one above.

Step 6: Graphing the Bi- Plot

```
bi_plot2017 = biplot(principal_component2017, choices = 1:2)
```

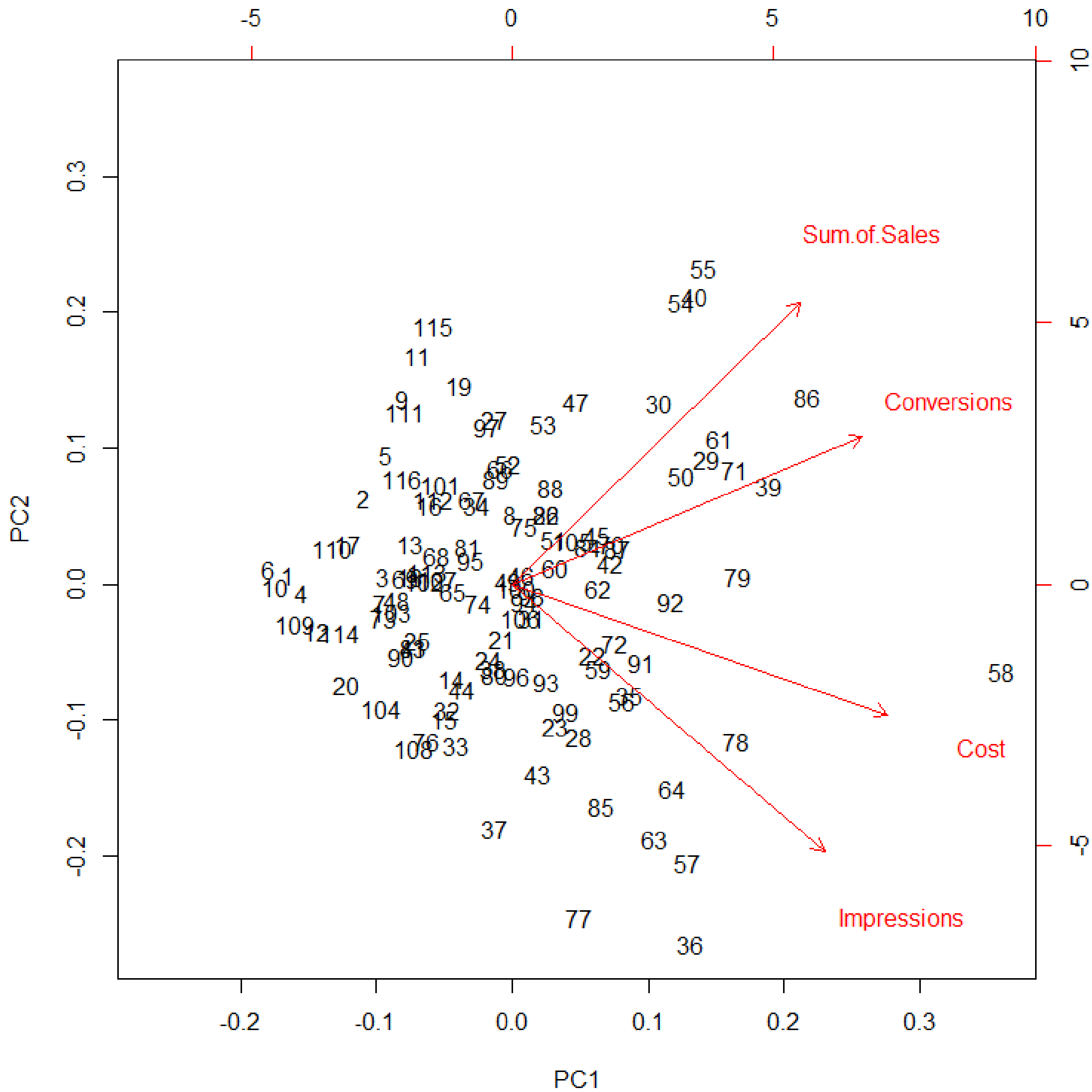


By using the bi plot function, we can plot the Principal Components that passed Kaiser's Rule. As a result, we are plotting PC1 and PC2 for the 2017 data. We do this by choosing the following:

- choices: Which Principal Components you want to plot. You will usually always plot the 1rst and 2nd Principal Components. Hence, we use the command 1:2.

To read the graph above, the vectors that are the closest to one another are the variables that have the highest correlation between each other. The farther a vector is from one another, the less correlation they have between each other. This is the 2017 data bi plot.

```
bi_plot2018 = biplot(principal_component2018, choices = 1:2)
```



By using the bi plot function, we can plot the Principal Components that passed Kaiser's Rule. As a result, we are plotting PC1 and PC2 for the 2018 data. We do this by choosing the following:

- choices: Which Principal Components you want to plot. You will usually always plot the 1rst and 2nd Principal Components. Hence, we use the command 1:2.

To read the graph above, the vectors that are the closest to one another are the variables that have the highest correlation between each other. The farther a vector is from one another, the less correlation they have between each other. This is the 2018 data bi plot.

Conclusion

In conclusion, by going through the following steps, we can get an in-depth analysis about all the moving parts when it comes to the variables in your data set. W can see which variables are highly correlated and therefore dependent with each other and see which variables are for the most part independent with each other. Principal Component Analysis lets the user look at the data in a form with less dimensions. For example, above we had 4 variables and therefore would need a graph in 4-D to look at how they interacted with each other however thanks to Principal Component Analysis we can now reduce the dimensionality to 2 dimensions and plot the data point onto a 2-D graph while still obtaining a very high explanatory power in terms of variability.