# post02-Allen-Chen

*Allen Chen*

*November 28, 2017*

## Reproducing Principal Component Analysis step-by-step Guide

This post uses the cereal dataset.

## Introduction:

In the morning of December 3rd, 2017, chef Bob wakes up hungry, and he wishes to make a bowl of cereal for himself. After shopping at the local organic supermarket for one hour and making his cereal in his newly remodeled kitchen for another hour, he is finally enjoying his yummy cereal. While devouring his bowl of cereal, he realized how great and delicious his cereal is and suddenly had a whim, thinking his cereal must be the best cereal in the world. “My recipe for mixing different proportions of various cereals must make me the best chef in the world!” he said.

Therefore, in order to convince himself that his cereal has the best overall rating in the cereal market, he gathered the cereal.csv file from https://www.kaggle.com/crawford/80-cereals/data below that shows 14 characteristics of 78 cereals, and their overall ratings, attempting to use cutting-edge and world famous machine learning model to predict his yet to be famous cereal’s rating.

Before training his cutting-edge and world famous machine learning model, he found out that there are too many variables to consider, and he was not sure which ones are more important than the other to accuratly predict the rating. In other word, how can he reduce the dimension of the feature space to best model the relationship between independent variables and avoid overfitting his model?

“Great Question”! he said to himself. So then he went to took a class with world famous Professor Sanchez. He learned to use -

Principal Component Analysis or PCA!

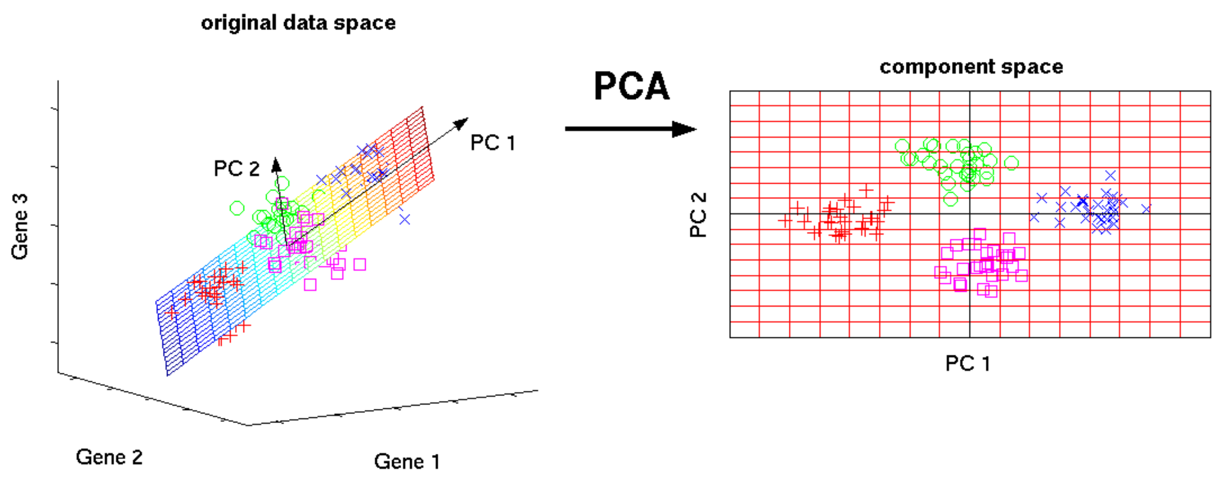
“Principal Component Analysis”, he thought. Since he is excited by this another world famous analysis, he did an extensive research. He found PCA is a way of identifying patterns in data and simultaneously compress the data to reduce the number of dimensions without losing much information. It is often used in image compression, feature extraction, speech recognition, text processing, and recommendation engines. This technique can allow him to create new independent variables through linear combinations of old variables. These new variables then have the order of how well they predict the dependent variable, so he can drop some of the least important ones while preserving some of the most important elements from those least important old independent variables!

Instead of using the PCA functions such as prcomp() and princomp(), he wants to implement his own PCA calculation to fully understand the concept within PCA.

Let’s start helping chef Bob perform Principal Component Analysis, and extract new important features and patterns from those known independent variables (mfr, type, calories…)!

After reading through this post, Chef Bob would be able to perform his own Principal Component Analysis manually and understand where all numbers come from and what they mean.

## Principal Component Analysis Step-by-Step Tutorial



This picture shows a three dimension data space that is reduced to two dimension component space

First, we important necessary packages and import the dataset cereal.csv

```
#load libraries
library(dplyr)
library(ggplot2)
library(scatterplot3d)

# import dataset cereal
cereals <- read.csv("../data/cereal.csv")
head(cereals)
```

```
##                        name mfr type calories protein fat sodium fiber
## 1                 100% Bran   N    C       70       4   1    130  10.0
## 2         100% Natural Bran   Q    C      120       3   5     15   2.0
## 3                  All-Bran   K    C       70       4   1    260   9.0
## 4 All-Bran with Extra Fiber   K    C       50       4   0    140  14.0
## 5            Almond Delight   R    C      110       2   2    200   1.0
## 6   Apple Cinnamon Cheerios   G    C      110       2   2    180   1.5
##   carbo sugars potass vitamins shelf weight cups   rating
## 1   5.0      6    280       25     3      1 0.33 68.40297
## 2   8.0      8    135        0     3      1 1.00 33.98368
## 3   7.0      5    320       25     3      1 0.33 59.42551
## 4   8.0      0    330       25     3      1 0.50 93.70491
## 5  14.0      8     -1       25     3      1 0.75 34.38484
## 6  10.5     10     70       25     1      1 0.75 29.50954
```

```
# Quick data dictionary:
# mfr: Macufacturer of cereal. A = American Home Food Products, N = Babisco
#                              G = General Mills, P = Post, Q = Quaker Oats
#                              K = Kelloggs, R = Ralston Purinia
# type: Cold or Hot
# shelf: display shelf (1, 2, or 3) counting from the floor
# weight: weight in ounces of one serving
# cups: number of cups in one serving
```

Since PCA only works well with continuous variables, we are not able to use variable mfr and type. We separate the dataset into independent variable set and dependent variable set.

```
# dataset with continuous variables only
cereals_x <- cereals %>% select(4:15) # 12 continuous variables: calories, protein, fat, sodium...
cereals_y <- cereals %>% select(16) # rating
dim(cereals_x)
```

```
## [1] 77 12
```

```
head(cereals_x)
```

```
##   calories protein fat sodium fiber carbo sugars potass vitamins shelf
## 1       70       4   1    130  10.0   5.0      6    280       25     3
## 2      120       3   5     15   2.0   8.0      8    135        0     3
## 3       70       4   1    260   9.0   7.0      5    320       25     3
## 4       50       4   0    140  14.0   8.0      0    330       25     3
## 5      110       2   2    200   1.0  14.0      8     -1       25     3
## 6      110       2   2    180   1.5  10.5     10     70       25     1
##   weight cups
## 1      1 0.33
## 2      1 1.00
## 3      1 0.33
## 4      1 0.50
## 5      1 0.75
## 6      1 0.75
```

Second, we subtract each column by their column mean.

```
col_means <- colMeans(cereals_x, n = 12)    # Get the mean of each column
adjusted_x <- cereals_x
for (i in 1:ncol(cereals_x)) {             # subtract the mean of mean of its respective column
  adjusted_x[,i] = adjusted_x[,i] - col_means[i]
}
dim(adjusted_x)
```

```
## [1] 77 12
```

```
head(adjusted_x)
```

```
##     calories    protein         fat     sodium      fiber      carbo
## 1 -36.883117  1.4545455 -0.01298701  -29.67532  7.8480519 -9.5974026
## 2  13.116883  0.4545455  3.98701299 -144.67532 -0.1519481 -6.5974026
## 3 -36.883117  1.4545455 -0.01298701  100.32468  6.8480519 -7.5974026
## 4 -56.883117  1.4545455 -1.01298701  -19.67532 11.8480519 -6.5974026
## 5   3.116883 -0.5454545  0.98701299   40.32468 -1.1519481 -0.5974026
## 6   3.116883 -0.5454545  0.98701299   20.32468 -0.6519481 -4.0974026
##       sugars    potass   vitamins      shelf      weight        cups
## 1 -0.9220779 183.92208  -3.246753  0.7922078 -0.02961039 -0.49103896
## 2  1.0779221  38.92208 -28.246753  0.7922078 -0.02961039  0.17896104
## 3 -1.9220779 223.92208  -3.246753  0.7922078 -0.02961039 -0.49103896
## 4 -6.9220779 233.92208  -3.246753  0.7922078 -0.02961039 -0.32103896
## 5  1.0779221 -97.07792  -3.246753  0.7922078 -0.02961039 -0.07103896
## 6  3.0779221 -26.07792  -3.246753 -1.2077922 -0.02961039 -0.07103896
```

Third, Obtain the Covariance Matrix

```
cov_matrix <- cov(adjusted_x) # use cov()
cov_matrix
```

```
##             calories     protein         fat       sodium        fiber
## calories 379.6308954  0.40669856  9.77785373  491.0799727 -13.62542720
## protein    0.4066986  1.19856459  0.22966507   -5.0179426   1.30550239
## fat        9.7778537  0.22966507  1.01298701   -0.4562543   0.04010595
## sodium   491.0799727 -5.01794258 -0.45625427 7027.8537252 -14.12106972
## fiber    -13.6254272  1.30550239  0.04010595  -14.1210697   5.68042379
## carbo     20.8996924 -0.61303828 -1.36970267  127.6965140  -3.63144224
## sugars    48.7012987 -1.60167464  1.21155161   37.8033151  -1.49589884
## potass   -92.5170882 42.87799043 13.86739576 -194.8427888 153.48537252
## vitamins 115.5160629  0.17942584 -0.70061517  677.0548530  -1.71693438
## shelf      1.5772386  0.12200957  0.22095010   -4.8658578   0.59037936
## weight     2.0408749  0.03561005  0.03250513    3.8926350   0.08866524
## cups       0.3953862 -0.06228469 -0.04119788    2.3345523  -0.28456784
##                  carbo      sugars      potass    vitamins       shelf
## calories   20.89969241 48.70129870  -92.517088 115.5160629  1.57723855
## protein    -0.61303828 -1.60167464   42.877990   0.1794258  0.12200957
## fat        -1.36970267  1.21155161   13.867396  -0.7006152  0.22095010
## sodium    127.69651401 37.80331511 -194.842789 677.0548530 -4.86585783
## fiber      -3.63144224 -1.49589884  153.485373  -1.7169344  0.59037936
## carbo      18.30946685 -6.30809979 -106.665584  24.6795967 -0.36261107
## sugars     -6.30809979 19.75700615    6.874573  12.4273753  0.37166781
## potass   -106.66558442  6.87457280 5081.809638  32.9673616 21.40464798
## vitamins   24.67959672 12.42737526   32.967362 499.1883117  5.56647300
## shelf      -0.36261107  0.37166781   21.404648   5.5664730  0.69309638
## weight      0.08701213  0.30141661    4.465689   1.0769395  0.02389781
## cups        0.36239747 -0.03347061   -8.215082   0.6676350 -0.06495557
##                weight         cups
## calories  2.040874915  0.395386193
## protein   0.035610048 -0.062284689
## fat       0.032505126 -0.041197881
## sodium    3.892634997  2.334552290
## fiber     0.088665243 -0.284567840
## carbo     0.087012133  0.362397471
## sugars    0.301416610 -0.033470608
## potass    4.465688653 -8.215082023
## vitamins  1.076939508  0.667634997
## shelf     0.023897813 -0.064955571
## weight    0.022643267 -0.006989064
## cups     -0.006989064  0.054156801
```

Fourth, we calculate the Eigenvectors and Eigenvalues of the above covariance matrix

```
eigen <- eigen(cov_matrix)
# Eigenvalues
eigen$values
```

```
##  [1] 7.156230e+03 5.071766e+03 4.666890e+02 3.113456e+02 2.150441e+01
##  [6] 5.512584e+00 7.654527e-01 6.461513e-01 4.309549e-01 2.821162e-01
## [11] 3.400767e-02 4.465695e-03
```

```
# EigenVectors
eigen$vectors
```

```
##                 [,1]          [,2]         [,3]          [,4]         [,5]
##  [1,] -0.0746837627  0.0092338161 -0.482406894  0.8603726948 -0.083011439
##  [2,]  0.0012633084 -0.0083246509 -0.001741869  0.0031279781 -0.072642567
##  [3,]  0.0001593141 -0.0026967177 -0.009008756  0.0295956608  0.036033395
##  [4,] -0.9871901312 -0.0955342086  0.126273421 -0.0134664199  0.012279802
##  [5,]  0.0041836160 -0.0298965572  0.014319306 -0.0254641660 -0.009049664
##  [6,] -0.0196581834  0.0185389547 -0.033388527  0.0042087347 -0.706709774
##  [7,] -0.0058068869 -0.0020434005 -0.065489694  0.1215746929  0.697171265
##  [8,]  0.0957818603 -0.9944893407  0.000783466  0.0201726029 -0.016269781
##  [9,] -0.1013080342 -0.0209816093 -0.863399720 -0.4927606254  0.022172749
## [10,]  0.0008636148 -0.0041314601 -0.013241766 -0.0027456023  0.004516472
## [11,] -0.0005141799 -0.0009501383 -0.003088643  0.0041714536 -0.001108534
## [12,] -0.0004467367  0.0015680354 -0.001053155 -0.0005863475 -0.006051574
##                [,6]         [,7]          [,8]         [,9]        [,10]
##  [1,]  0.100998652 -0.036750158  0.0059353694  0.026982957  0.045996544
##  [2,]  0.218888031 -0.175421194 -0.4816554125 -0.765865598 -0.311448442
##  [3,]  0.209836721  0.388255122  0.3813080789  0.057088361 -0.807435155
##  [4,]  0.006818838  0.001106089  0.0006866356 -0.001363284  0.001295309
##  [5,]  0.028130051 -0.873041187  0.4351740704  0.019151153 -0.208766606
##  [6,] -0.676326972  0.037866695  0.0257813003 -0.077034265 -0.181474494
##  [7,] -0.662415199 -0.016362058 -0.0423412862 -0.153253724 -0.174367090
##  [8,] -0.014623399  0.026101423 -0.0121955349  0.007328356  0.006813127
##  [9,]  0.018814448 -0.001066006 -0.0066780766  0.009988282 -0.005440903
## [10,]  0.001473117  0.226058366  0.6523157955 -0.615344562  0.372381468
## [11,] -0.009773636 -0.030575320 -0.0119984405  0.004213834  0.032808398
## [12,] -0.011851806  0.022614048 -0.0699241989  0.029205485 -0.046799969
##               [,11]         [,12]
##  [1,]  3.796953e-03 -6.624120e-03
##  [2,] -2.011837e-02  2.801031e-03
##  [3,] -2.237071e-02  4.309102e-02
##  [4,]  1.178005e-04 -5.779688e-05
##  [5,]  4.134405e-02 -1.118271e-02
##  [6,] -1.770247e-02 -1.126371e-03
##  [7,] -9.834927e-03 -1.820917e-03
##  [8,]  6.281142e-05 -7.392296e-04
##  [9,] -1.809861e-03 -5.926087e-04
## [10,]  7.564029e-02  1.095949e-02
## [11,] -7.685925e-02  9.958890e-01
## [12,]  9.926373e-01  7.775574e-02
```

---

what do the eigenvalues and eigenvectors mean? Greaaaat Question!

It is very difficult to explain what those eigen values and eigenvectors mean since the eigenvector has a high dimension, thus making graphical representation of how eigenvectors represent the relationship between the points in the adjusted matrix difficult.

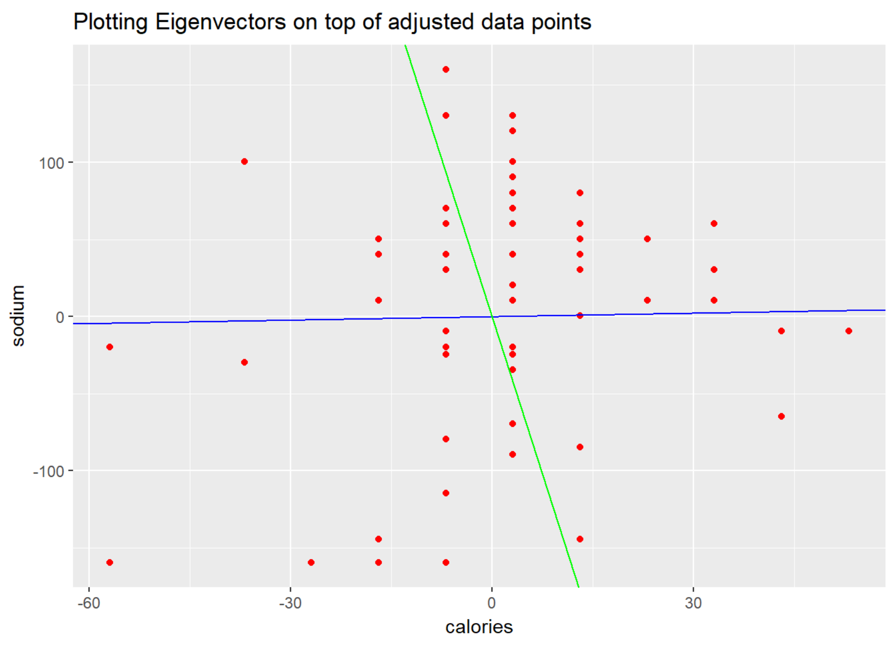
Therefore, pretend there are only two features in our cereal matrix,

```
two_features <- cereals_x[,c(1,4)]
mean_calories <- mean(two_features$calories)
mean_sodium <- mean(two_features$sodium)
adjusted_two_features <- two_features          # two features with column mean subtracted
adjusted_two_features[,1] <- adjusted_two_features[,1] - mean_calories
adjusted_two_features[,2] <- adjusted_two_features[,2] - mean_sodium
cov_two_features <- cov(adjusted_two_features) # covariance matrix of two features
eig_two_features <- eigen(cov_two_features)
# calculate slope as ratio (first column)
eig_two_features$slopes[1] <- eig_two_features$vectors[1,1]/eig_two_features$vectors[2,1]
# calculate slope as ratio (second column)
eig_two_features$slopes[2] <- eig_two_features$vectors[1,2]/eig_two_features$vectors[2,2]
eig_two_features                               # Eigenvectors and Eigenvalues are shown below
```

```
## eigen() decomposition
## $values
## [1] 7063.9322  343.5524
##
## $vectors
##            [,1]        [,2]
## [1,] 0.07327019 -0.99731213
## [2,] 0.99731213  0.07327019
##
## $slopes
## [1]   0.07346766 -13.61143133
```

Let’s plot the two-features adjusted data with eigenvectors laying on top.

```
ggplot(data = adjusted_two_features, aes(x = calories, y = sodium)) +
  geom_point(color = "red") +
  geom_abline(intercept = 0, slope = eig_two_features$slopes[1], colour = "blue") + # plot pc1
  geom_abline(intercept = 0, slope = eig_two_features$slopes[2], colour = "green")+ # plot pc2
  ggtitle("Plotting Eigenvectors on top of adjusted data points")
```



So what does this plot, eigenvectors and eigenvalues tell us? From the covariance matrix, the covariance is 491.08, which tells us that calories and sodium move together, but they have a correlation of 0.3, which means they have a weak correlation, sodium would only increase at the rate of 0.3 for every calorie increase.

The whole point of using Eigenvectors is to capture the underlying patterns or relationship between variables, and we can see the blue line basically shows us the relationship between sodium and calories as I just described. Another way to look at it is that the blue line looks like the line of best fit.

The green line shows us a less important relationship between sodium and calories as we can see that this line does not capture the relationship between sodium and calories.

Now we know about the eigenvectors help us extract pattern within variables. We can now look at Eigenvalues. We already know that the dimension or the number of features equal to the number of eigenvectors it has, and this is also true for eigenvalues, meaning one eigenvector corresponds to one eigenvalue.

Here in my opinion is the key behind Principal Component Analysis:

An eigenvalue is a number telling us how much variance there is in the data in that direction or eigenvector. In the two variables example, the blue and green lines are telling us how spread out the data is on the line. Therefore, higher the variance means higher the eigenvalues, further telling us the eigenvector with the highest eigenvalue is the principal component.

---

Let's go back to the high dimension case with 12 features!

```
# This finds the principal component in our eigenvector
prin_comp <- max(eigen$values)
prin_comp
```

```
## [1] 7156.23
```

Our Principle Component is the eigenvector with an eigenvalue of 7156.23! so it provides the main underlying structure when it comes to finding the relationship between our independent variables or features.

Fifth, Let's choose our components and form our feature vector (a matrix)

```
#let's revisit our eigenvalues and eigenvectors for 12 features
eigen
```

```
## eigen() decomposition
## $values
##  [1] 7.156230e+03 5.071766e+03 4.666890e+02 3.113456e+02 2.150441e+01
##  [6] 5.512584e+00 7.654527e-01 6.461513e-01 4.309549e-01 2.821162e-01
## [11] 3.400767e-02 4.465695e-03
##
## $vectors
##                [,1]          [,2]         [,3]          [,4]         [,5]
##  [1,] -0.0746837627  0.0092338161 -0.482406894  0.8603726948 -0.083011439
##  [2,]  0.0012633084 -0.0083246509 -0.001741869  0.0031279781 -0.072642567
##  [3,]  0.0001593141 -0.0026967177 -0.009008756  0.0295956608  0.036033395
##  [4,] -0.9871901312 -0.0955342086  0.126273421 -0.0134664199  0.012279802
##  [5,]  0.0041836160 -0.0298965572  0.014319306 -0.0254641660 -0.009049664
##  [6,] -0.0196581834  0.0185389547 -0.033388527  0.0042087347 -0.706709774
##  [7,] -0.0058068869 -0.0020434005 -0.065489694  0.1215746929  0.697171265
##  [8,]  0.0957818603 -0.9944893407  0.000783466  0.0201726029 -0.016269781
##  [9,] -0.1013080342 -0.0209816093 -0.863399720 -0.4927606254  0.022172749
## [10,]  0.0008636148 -0.0041314601 -0.013241766 -0.0027456023  0.004516472
## [11,] -0.0005141799 -0.0009501383 -0.003088643  0.0041714536 -0.001108534
## [12,] -0.0004467367  0.0015680354 -0.001053155 -0.0005863475 -0.006051574
##               [,6]         [,7]          [,8]         [,9]        [,10]
##  [1,]  0.100998652 -0.036750158  0.0059353694  0.026982957  0.045996544
##  [2,]  0.218888031 -0.175421194 -0.4816554125 -0.765865598 -0.311448442
##  [3,]  0.209836721  0.388255122  0.3813080789  0.057088361 -0.807435155
##  [4,]  0.006818838  0.001106089  0.0006866356 -0.001363284  0.001295309
##  [5,]  0.028130051 -0.873041187  0.4351740704  0.019151153 -0.208766606
##  [6,] -0.676326972  0.037866695  0.0257813003 -0.077034265 -0.181474494
##  [7,] -0.662415199 -0.016362058 -0.0423412862 -0.153253724 -0.174367090
##  [8,] -0.014623399  0.026101423 -0.0121955349  0.007328356  0.006813127
##  [9,]  0.018814448 -0.001066006 -0.0066780766  0.009988282 -0.005440903
## [10,]  0.001473117  0.226058366  0.6523157955 -0.615344562  0.372381468
## [11,] -0.009773636 -0.030575320 -0.0119984405  0.004213834  0.032808398
## [12,] -0.011851806  0.022614048 -0.0699241989  0.029205485 -0.046799969
##               [,11]         [,12]
##  [1,]  3.796953e-03 -6.624120e-03
##  [2,] -2.011837e-02  2.801031e-03
##  [3,] -2.237071e-02  4.309102e-02
##  [4,]  1.178005e-04 -5.779688e-05
##  [5,]  4.134405e-02 -1.118271e-02
##  [6,] -1.770247e-02 -1.126371e-03
##  [7,] -9.834927e-03 -1.820917e-03
##  [8,]  6.281142e-05 -7.392296e-04
##  [9,] -1.809861e-03 -5.926087e-04
## [10,]  7.564029e-02  1.095949e-02
## [11,] -7.685925e-02  9.958890e-01
## [12,]  9.926373e-01  7.775574e-02
```

Conveniently, the eigenvalues and eigenvectors are already sorted for us, and as we previously discussed, this basically shows us the ranking of eigenvectors in term of importance.

Therefore, if we desire to reduce the dimension of our features, instead of feature elimination by completely taking out unwanted features from the cereals_x matrix, we can take out the last few columns of eigen$vectors. By taking only the important eigenvectors, we are performng feature extraction, and this basically preserve all the important relationship between all features when reducing the dimension.

```
# Say we only want the top three eigenvectors or new independent variables
feature_vector <- eigen$vectors[, 1:3]
feature_vector
```

```
##                [,1]          [,2]         [,3]
##  [1,] -0.0746837627  0.0092338161 -0.482406894
##  [2,]  0.0012633084 -0.0083246509 -0.001741869
##  [3,]  0.0001593141 -0.0026967177 -0.009008756
##  [4,] -0.9871901312 -0.0955342086  0.126273421
##  [5,]  0.0041836160 -0.0298965572  0.014319306
##  [6,] -0.0196581834  0.0185389547 -0.033388527
##  [7,] -0.0058068869 -0.0020434005 -0.065489694
##  [8,]  0.0957818603 -0.9944893407  0.000783466
##  [9,] -0.1013080342 -0.0209816093 -0.863399720
## [10,]  0.0008636148 -0.0041314601 -0.013241766
## [11,] -0.0005141799 -0.0009501383 -0.003088643
## [12,] -0.0004467367  0.0015680354 -0.001053155
```

Sixth, Derive our features dataset or “new independent variables”

Now we have our feature vector, so let’s construct our dimension-reduced “new independent variables” that can be used to train Chef Bob’s machine learning model. The formula to derive our new feature dataset is the dot product of our transposed feature vector from above and the transposed adjusted dataset with mean zero. The result is a 3 X 77 matrix, so taking the transpose of that would return us a 77 X 3 matrix, which is what we want!

We have reduced from 12 features to 3 features! Chef Bob did it!

```
# Derive our new features dataset
new_cereals_x <- t(t(feature_vector) %*% t(adjusted_x))
dim(new_cereals_x)
```

```
## [1] 77  3
```

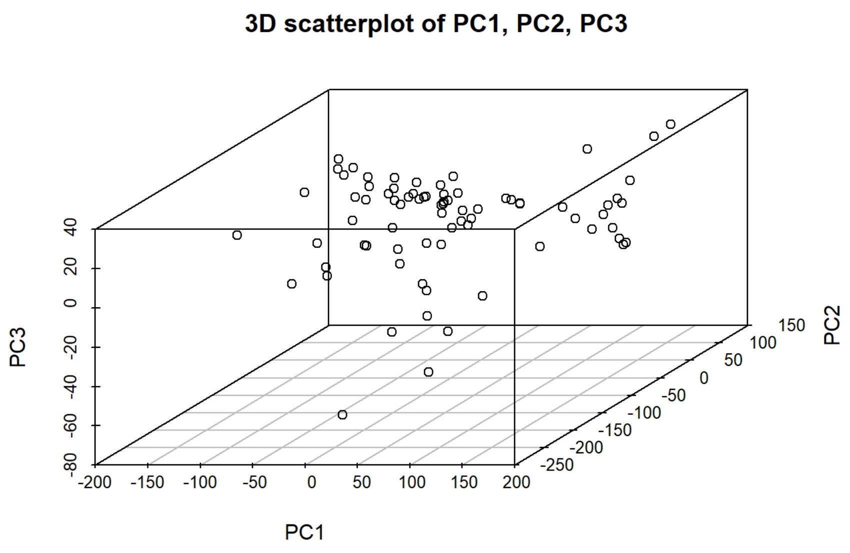
```
head(new_cereals_x)
```

```
##           [,1]       [,2]        [,3]
## [1,]  50.22469 -180.77275 17.47371777
## [2,] 148.55671  -24.30984 -0.07737806
## [3,] -74.31645 -232.90275 33.90499443
## [4,]  46.62792 -231.68598 28.78264268
## [5,] -49.00942   92.80716  6.23011028
## [6,] -22.40755   24.03344  3.77979160
```

## Graph the New Feature Dataset

The graph below shows the underlying relationship across all features, and the values of those points tell us exactly where the trend lines the data points sit. In other words, those points are clearly shown that can be below or above the axes lines since the coordinates of this graph is in term of the eigenvectors.

```
new_cereals_x <- data.frame(new_cereals_x)      # turning a matrix into a dataframe
names(new_cereals_x) <- c("PC1", "PC2", "PC3")  # give it a name
scatterplot3d(new_cereals_x, main = "3D scatterplot of PC1, PC2, PC3")
```



## Double check with the built-in PCA function

How does our new feature dataset compare to the built-in PCA function, namely prcomp()?

Let’s see!

```
pca_prcomp <- prcomp(cereals_x)        # apply prcomp() to get
pca_prcomp_three <- pca_prcomp$x[,1:3]
dim(pca_prcomp_three)
```

```
## [1] 77  3
```

```
head(pca_prcomp_three)
```

```
##             PC1        PC2          PC3
## [1,]  -50.22469 -180.77275 -17.47371777
## [2,] -148.55671  -24.30984   0.07737806
## [3,]   74.31645 -232.90275 -33.90499443
## [4,]  -46.62792 -231.68598 -28.78264268
## [5,]   49.00942   92.80716  -6.23011028
## [6,]   22.40755   24.03344  -3.77979160
```

They are the same as our newly created feature dataset (new_cereals_x)!

After Chef Bob successfully computed PCA, he trained his world famous machine learning model. He then input the feature of his cereal, and he received a rating of 99.99999! With this data, he confidently opened the world famous cereal restaurant, and he naturally becomes the world famous Chef Bob who makes world famous cereal.

## Summary

Principal Component Analysis is a way to extract underlying patterns between features or indepenedent variables to form a new feature dataset that best represent their relationships with decreasing importance through the use of standardization, covariance matrix, eigenvalues and eigenvectors. This allows us to remove some least important “new variables”, thus achieving the goal of dimentionality reduction. Because we lose so little information by dropping the least important columns, this can apply to fields like imaging compression and text processing.

This leads us to the take-home message: This post provides a step-by-step guide that shows readers how to perform Principal Component Analysis with reproducible codes, detailed explanation of linear algebra techniques and the intuition behind dimentionality reduction.

## References

1. https://www.kaggle.com/crawford/80-cereals/data

- The cereal dataset can be downloaded from the kaggle website.

2. https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

- This article written by Matt provided a much more detailed calculation and performance of Principal Components Analysis

3. https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/

- This article by George gave a good intuition behind the process of obtaing principal components.

4. http://www.gastonsanchez.com/visually-enforced/how-to/2012/06/17/PCA-in-R/

- This article by Prof. Sanchez introduced us many of useful PCA functions to use in R.

5. https://klevas.mif.vu.lt/~tomukas/Knygos/principal_components.pdf

- This short book written by Lindsay provided us a more detailed tutorial on how to perform PCA.

6. https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/

- This article has many great graphs on PCA.

7. https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues

- This answer to the question on PCA at stackexchange gives a good intuition and explanation on PCA in a langauge that everyone can understand.

8. https://rpubs.com/aaronsc32/eigenvalues-eigenvectors-r

- This article teaches us how to obtain eigenvectors and eigenvalues in R with detailed mathematical explanations.