Dimensionality Reduction

Nick Kallfa

January 25, 2018

Dimensionality Reduction

An issue that we often deal with when clustering data is the curse of dimensionality. Basically, this means that when you have data with a lot of columns (i.e. high dimensionality) weird things happen. This is particularly problematic when clustering because we are trying to find groups of similar objects and distance is a metric commonly used to measure similarity. In other words, objects are more similar to one another if they are closer to each other. However, in very high dimensions, distance is not a good metric to use for similarity. We could try using other distance metrics (correlation-based distance), but this can get computationally expensive. Prior to clustering dimensionality reduction is often performed to extract the most interesting pieces of information from your data. After reducing the dimensions of the data we then cluster in the reduced space.

In this notebook I wanted to explore performing dimensionality reduction using a couple different techniques on some sample data sets and then we will reduce the dimensionality of the government contracting data I have been working with. First, we use principal component analysis on the Iris data set. The Iris data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

```
df = iris
Species = data.frame(Species = df$Species)
df = df[,c(1,2,3,4)]
pc = prcomp(df, center = TRUE, scale. = TRUE)
pca = data.frame(pc$x)
pca$Species = Species$Species
```

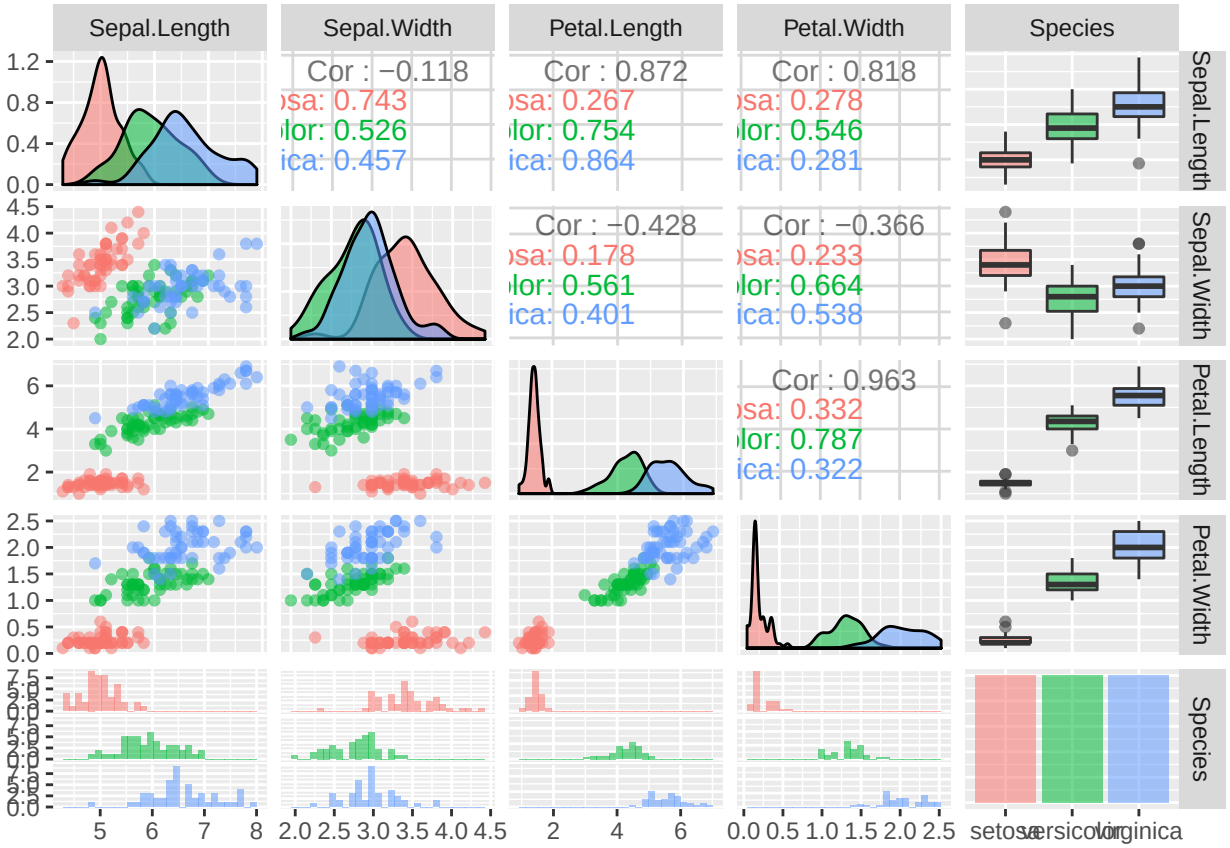
Principal Components for the Iris Data Set

Principal component analysis (PCA) is a standard dimensionality reduction technique. It's usually the first method for dimensionality reduction you'll learn about in a machine learning course. PCA finds the most interesting pieces of information in your data by finding the directions in your data that (1) vary the most and (2) are perpendicular to each other. Intuitively, if you have features (columns) in your data that are highly correlated with each other PCA will condense that information into a much smaller set of features.

We start by plotting each of the four features against one another

```
ggpairs(iris, aes(colour = Species, alpha = 0.4))
```

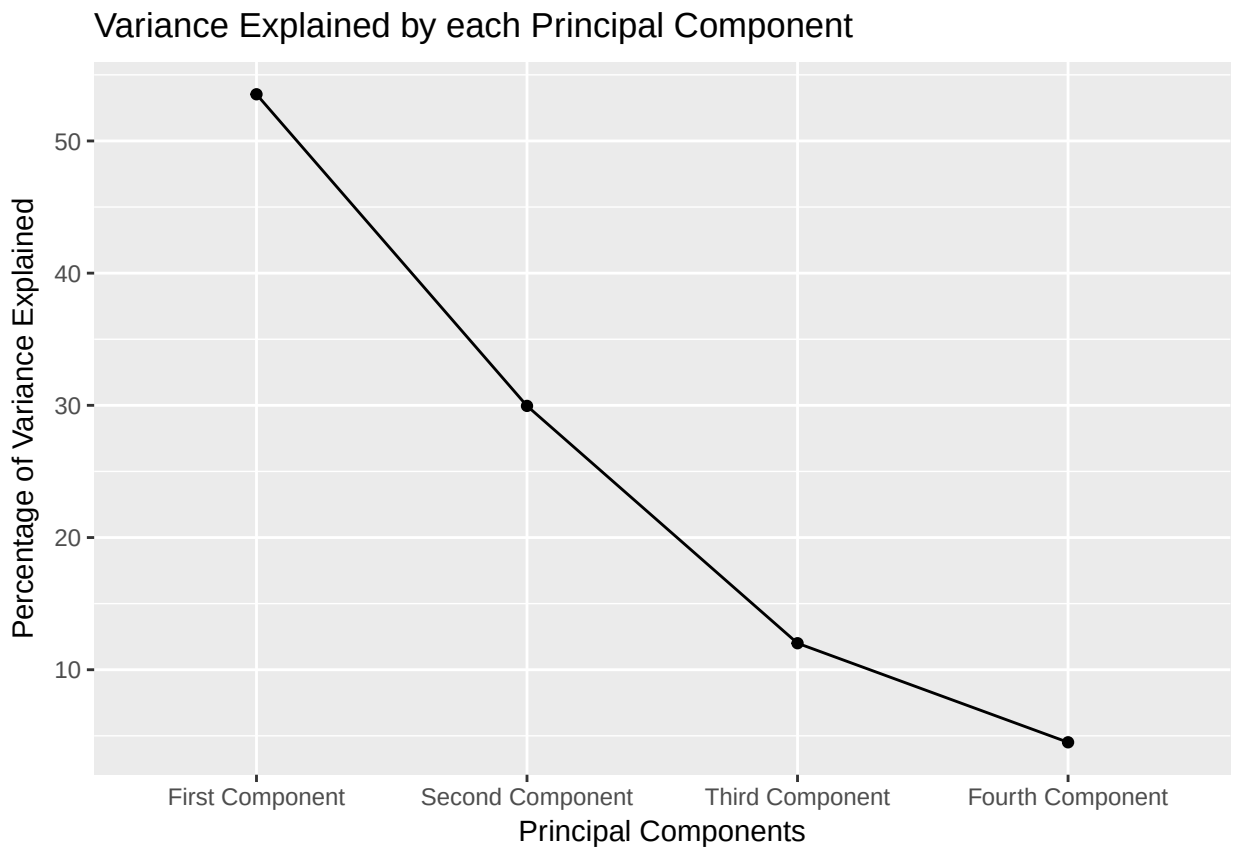
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



If we look at the second image in the first column we have sepal length plotted against sepal width. The points in red which correspond to the setosa species are fairly well-separated from the blue (versicolor species) and green (virginica species) points. However, the blue and green points overlap with one another. It looks like the blue and green points are better separated when using petal width (see across the fourth row) as one of our plotting variables.

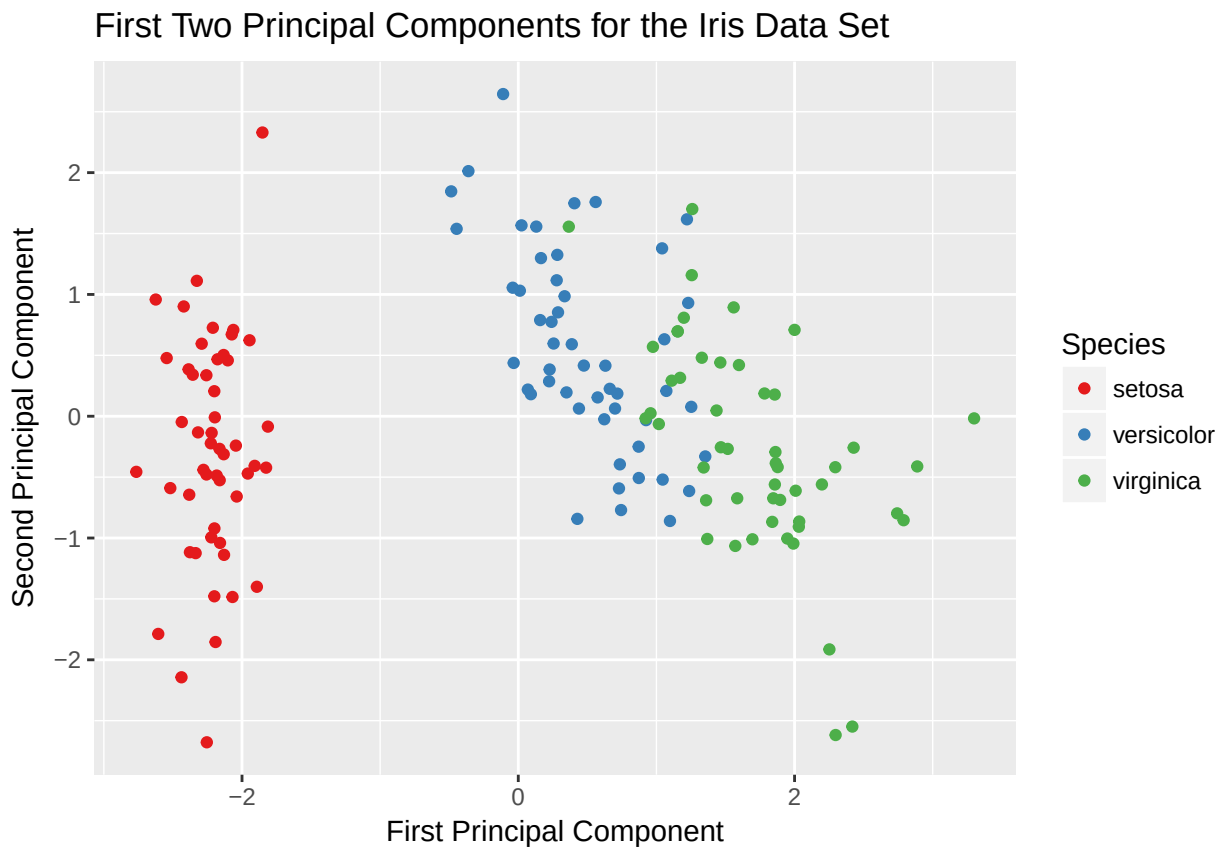
Now we can start looking at the data obtained after performing PCA. First, we can see just how much variance is explained by each principal component. This tells us how much of the interesting information is contained in each component

```
variance_explained = data.frame(var = pc$sdev/sum(pc$sdev)*100)
variance_explained$component = factor(c("First Component", "Second Component", "Third Component", "Four
                                        levels = c("First Component", "Second Component", "Third Componen
ggplot(data = variance_explained) +
  geom_point(mapping = aes(x = component, y = var)) +
  geom_line(mapping = aes(x = 1:4, y = var)) +
  labs(title = "Variance Explained by each Principal Component",
       y = "Percentage of Variance Explained",
       x = "Principal Components")
```



So we see that in just the first principal component alone it explains over 50% of the variance! The second principal component explains 30% of the variance. So just in the first two components we have over 80% of the variance of the original data set contained within it. We often see this type of behavior when performing PCA and other dimensionality reduction techniques. Since the first two components explain over 80% of the variance of the original data set we can plot the first two components against each other to see what it looks like. We will also color each point according to the species of Iris flower it is associated with

```
ggplot(pca) +
  geom_point(mapping = aes(x = PC1, y = PC2, colour = Species)) +
  labs(title = "First Two Principal Components for the Iris Data Set",
       x = "First Principal Component",
       y = "Second Principal Component") +
  scale_colour_brewer(palette = "Set1")
```



Above we see each sample in the Iris data set plotted. Along the x-axis is the values from the first principal component and along the y-axis are the values from the second principal component. Each point is colored according to what species of Iris it is.

So we see that in just the first two principal components we get fairly good separation between the groups. The red points are clearly separated from the green and the blue. The green and blue still overlap somewhat. That's what PCA looks like on a small, exemplary data set.

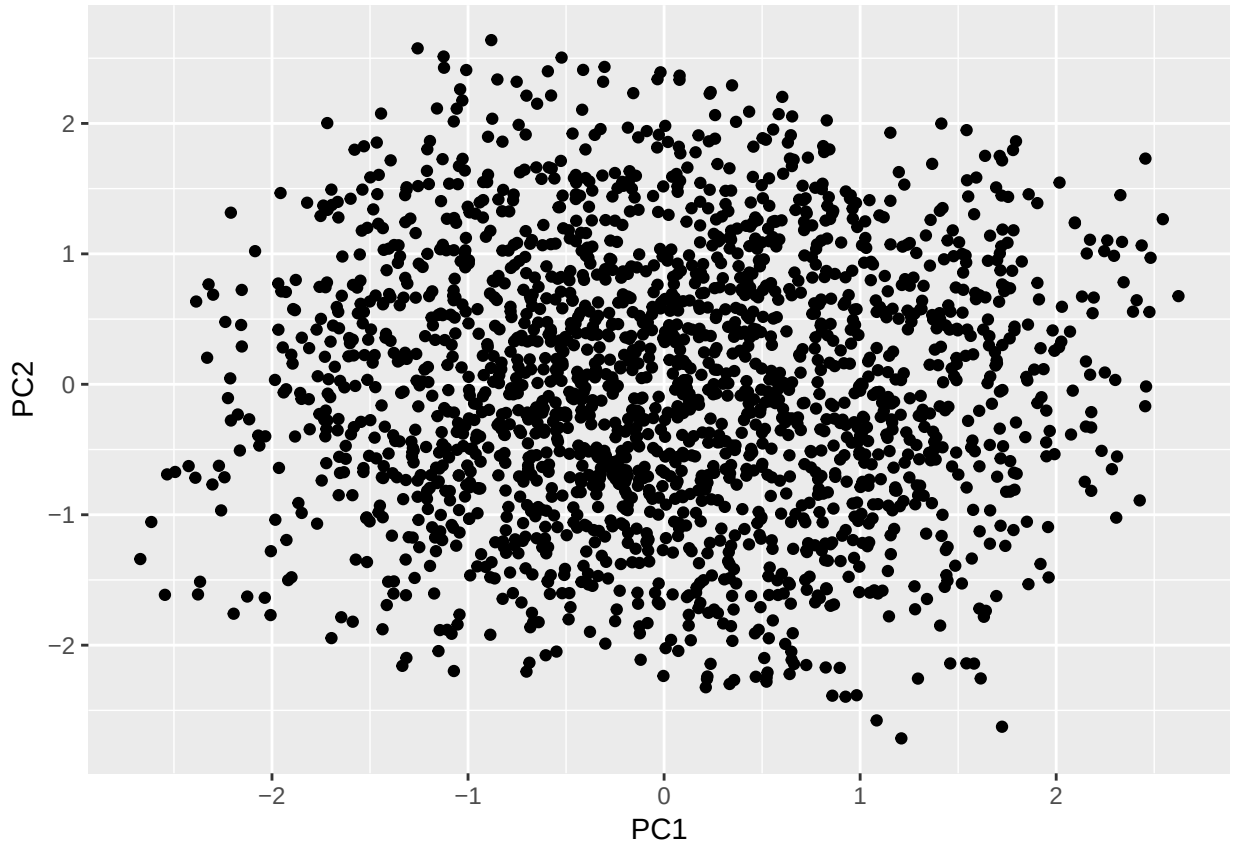
Random (uniformly) Generated Data

Next we generate completely random data and look at what PCA gives for that. We create a data set consisting of 4 columns. Each column contains 2000 numbers randomly chosen between 5,000 and 1,000,000. Then we perform PCA and plot the first two components against each other.

PCA on Random Data

```
set.seed(10)

df_random = data.frame(X1 = runif(2000, min = 5000, max = 1000000),
                       X2 = runif(2000, min = 5000, max = 1000000),
                       X3 = runif(2000, min = 5000, max = 1000000),
                       X4 = runif(2000, min = 5000, max = 1000000))

pc_random = prcomp(x = df_random, center = TRUE, scale. = TRUE)
```

```r
pc_random = data.frame(pc_random$x)

ggplot(data = pc_random) +
  geom_point(mapping = aes(x = PC1, y = PC2))
```



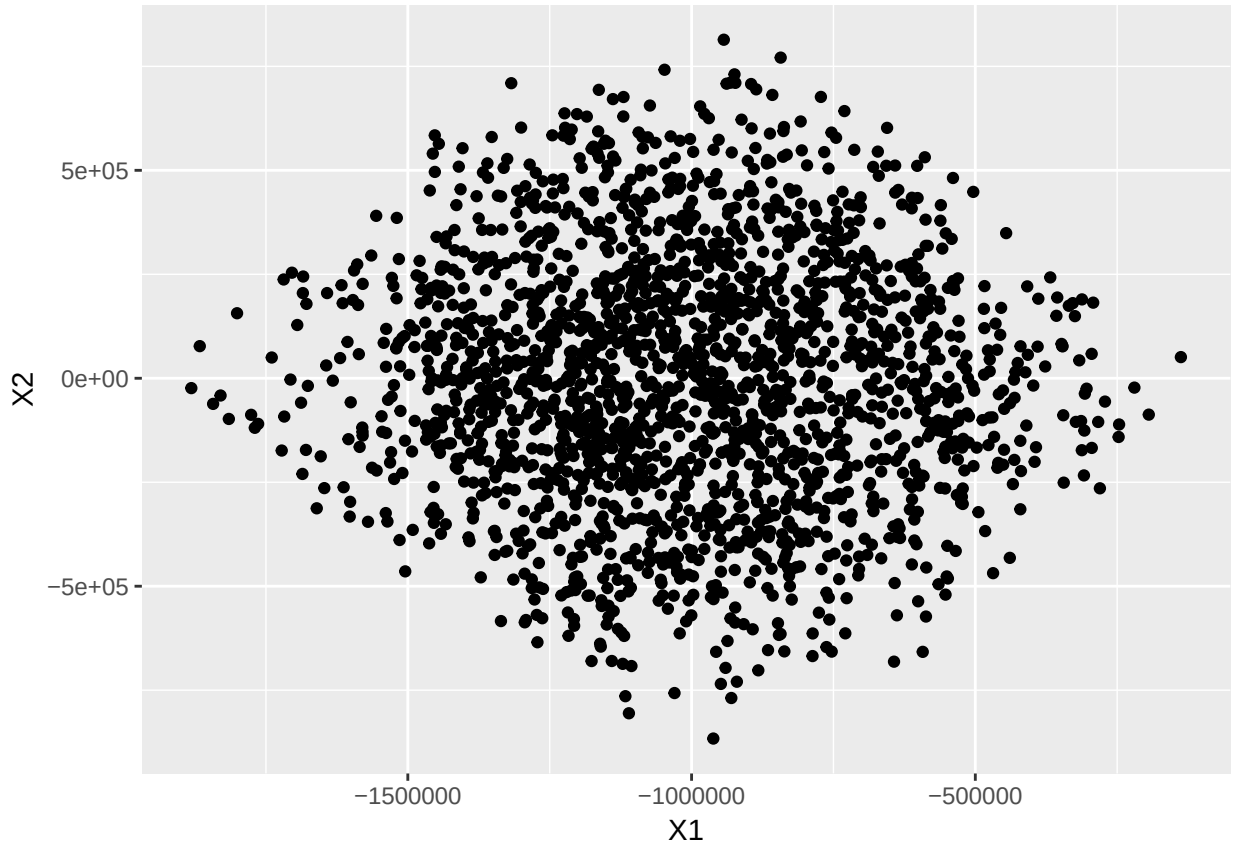
Since the data is random we wouldn't expect to see any groups emerge out of this, but this is what you'd expect to see if the data you have is uniformly random. This gives us a baseline understanding for interpreting other PCA component plots. Since clustering is an unsupervised technique (i.e. you have no idea if clusters exist in your data) we don't really know what to look for. This gives us an idea of what we'd see if there truly were no groups in the data.

SVD on Random Data

PCA is not the only method for dimensionality reduction. We also have singular value decomposition (SVD) which is a related technique. I'm not going to get into the mathematics of it, but you can think of it the same way you think of PCA: it condenses your data into its most interesting pieces of information. Just as when we performed PCA we obtained principal components, when we perform SVD we obtain singular vectors.

Let's calculate the SVD of the random data and plot the fist two singular vectors against each another.

```r
m = as.matrix(df_random)
svd = svd(x = m, nu = 4, nv = 4)
svd_components = data.frame(svd$u %*% diag(svd$d))

ggplot(data = svd_components) +
  geom_point(mapping = aes(x = X1, y = X2))
```



Not surprisingly ,it looks like the PCA plot. This is what you might see if you perform SVD on uniform random variables.

Synthetic Data with Groups

Next, we will generate a data set that does have groups. We create a data set that has 10,000 rows and 10 columns. Each group is created by randomly choosing a number between 5,000 and 1,000,000 for 2 of the columns. We set the rest of the 8 columns in that group to 0.

This data was generated on purpose because it has a couple special characteristics:

1. **Sparsity:** I wanted to simulate sparse data (data with many zeroes) so that it would be similar to the FPDS data I am working with
2. **Correlation between columns:** The FPDS data I have is highly correlated among its columns. A company engaged in aircraft manufacturing is probably also involved in manufacturing screws/bolts. Dimensionality reduction will condense this type of correlated information.

By simulating data that is sparse and correlated I can get an idea of what to expect with real-life data.

```
set.seed(10)

First_Group = data.frame(X1 = runif(2000, min = 5000, max = 1000000),
                         X2 = runif(2000, min = 5000, max = 1000000),
                         X3 = rep(0, 2000),
                         X4 = rep(0, 2000),
```

```
                          X5 = rep(0, 2000),
                          X6 = rep(0, 2000),
                          X7 = rep(0, 2000),
                          X8 = rep(0, 2000),
                          X9 = rep(0, 2000),
                          X10 = rep(0, 2000))
Second_Group = data.frame(X1 = rep(0, 2000),
                          X2 = rep(0, 2000),
                          X3 = runif(2000, min = 5000, max = 1000000),
                          X4 = runif(2000, min = 5000, max = 1000000),
                          X5 = rep(0, 2000),
                          X6 = rep(0, 2000),
                          X7 = rep(0, 2000),
                          X8 = rep(0, 2000),
                          X9 = rep(0, 2000),
                          X10 = rep(0, 2000))
Third_Group = data.frame(X1 = rep(0, 2000),
                         X2 = rep(0, 2000),
                         X3 = rep(0, 2000),
                         X4 = rep(0, 2000),
                         X5 = runif(2000, min = 5000, max = 1000000),
                         X6 = runif(2000, min = 5000, max = 1000000),
                         X7 = rep(0, 2000),
                         X8 = rep(0, 2000),
                         X9 = rep(0, 2000),
                         X10 = rep(0, 2000))
Fourth_Group = data.frame(X1 = rep(0, 2000),
                          X2 = rep(0, 2000),
                          X3 = rep(0, 2000),
                          X4 = rep(0, 2000),
                          X5 = rep(0, 2000),
                          X6 = rep(0, 2000),
                          X7 = runif(2000, min = 5000, max = 1000000),
                          X8 = runif(2000, min = 5000, max = 1000000),
                          X9 = rep(0, 2000),
                          X10 = rep(0, 2000))
Fifth_Group = data.frame(X1 = rep(0, 2000),
                         X2 = rep(0, 2000),
                         X3 = rep(0, 2000),
                         X4 = rep(0, 2000),
                         X5 = rep(0, 2000),
                         X6 = rep(0, 2000),
                         X7 = rep(0, 2000),
                         X8 = rep(0, 2000),
                         X9 = runif(2000, min = 5000, max = 1000000),
                         X10 = runif(2000, min = 5000, max = 1000000))

df = rbind(First_Group, Second_Group, Third_Group, Fourth_Group, Fifth_Group)

# Label for groups
Group1 = data.frame(Group = rep(1, 2000))
Group2 = data.frame(Group = rep(2, 2000))
Group3 = data.frame(Group = rep(3, 2000))
```

```
Group4 = data.frame(Group = rep(4, 2000))
Group5 = data.frame(Group = rep(5, 2000))
Groups = rbind(Group1, Group2, Group3, Group4, Group5)
df$Group = factor(Groups$Group)
```

Let's randomly reshuffle rows of the data and take a look at the top columns so we know what the data look like.

```
df = df[sample(nrow(df), nrow(df)), ]
rownames(df) = 1:nrow(df)
head(df,10)
```

```
##          X1       X2       X3       X4       X5        X6       X7
## 1      0.00      0.0      0.0      0.0      0.0      0.00      0.0
## 2      0.00      0.0      0.0      0.0 928731.9  76217.19      0.0
## 3      0.00      0.0      0.0      0.0 662910.2 985921.16      0.0
## 4      0.00      0.0 427341.0 411783.0      0.0      0.00      0.0
## 5      0.00      0.0      0.0      0.0 626177.6 716702.97      0.0
## 6      0.00      0.0      0.0      0.0 755155.7 471719.01      0.0
## 7      0.00      0.0 991421.1 861914.5      0.0      0.00      0.0
## 8      0.00      0.0      0.0      0.0 275367.9 129278.34      0.0
## 9  92983.35 603988.1      0.0      0.0      0.0      0.00      0.0
## 10     0.00      0.0      0.0      0.0      0.0      0.00 201856.6
##          X8     X9      X10 Group
## 1       0.0 844277 689509.7     5
## 2       0.0      0      0.0     3
## 3       0.0      0      0.0     3
## 4       0.0      0      0.0     2
## 5       0.0      0      0.0     3
## 6       0.0      0      0.0     3
## 7       0.0      0      0.0     2
## 8       0.0      0      0.0     3
## 9       0.0      0      0.0     1
## 10 594112.1      0      0.0     4
```

In the view above we can see the groups. Each row contains values only in a couple of columns. The rest of the columns are zero.

Now that we have created this data let's perform PCA and plot some of the components against each other. We do PCA in a couple different ways

1. No centering and no scaling
2. With centering and scaling

Centering and scaling means we transform our data by subtracting the column mean from each of the values and divide by the column's standard deviation. This is a standard technique used to make sure all the columns of the data are on the same scale so none overpower any others.

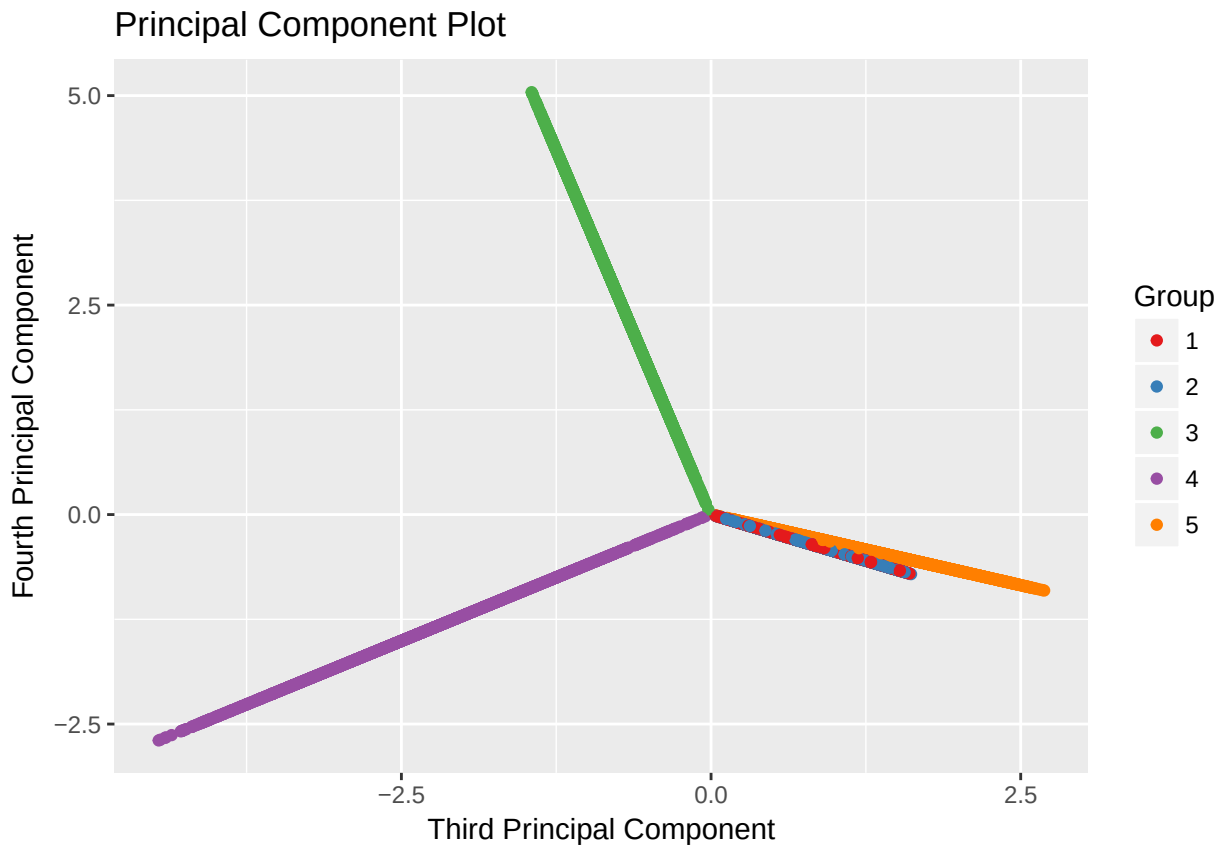
PCA

```
data_for_pca = df[,1:(ncol(df) - 1)]

pc_center_scale = prcomp(data_for_pca, center = TRUE, scale. = TRUE)
var_explained_center_scale = data.frame(var = pc_center_scale$sdev/sum(pc_center_scale$sdev))
pc_center_scale = data.frame(pc_center_scale$x)
```

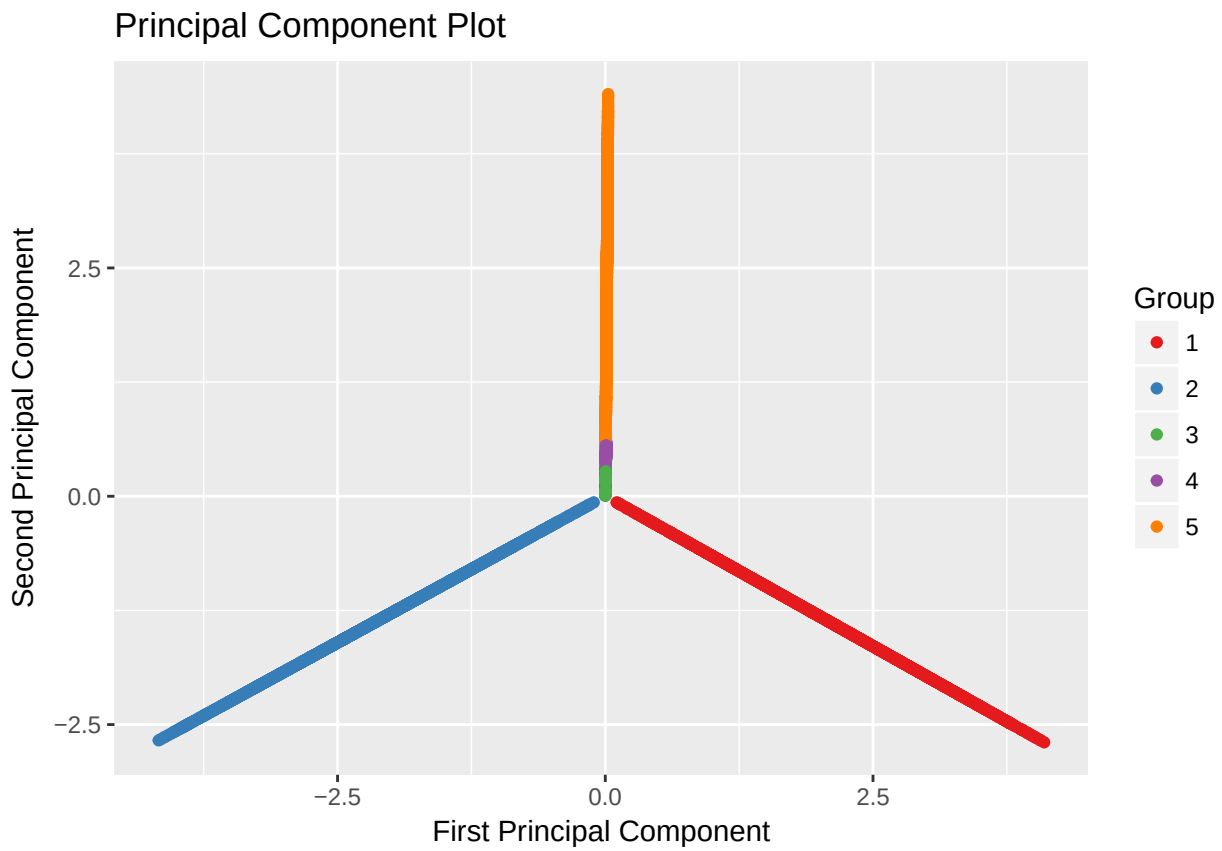
```r
pc_center_scale$Group = df$Group

pc_no_center_no_scale = prcomp(data_for_pca, center = TRUE, scale. = TRUE)
var_explained_no_center_no_scale = data.frame(var = pc_no_center_no_scale $sdev/sum(pc_no_center_no_sca
pc_no_center_no_scale = data.frame(pc_no_center_no_scale$x)
pc_no_center_no_scale$Group = df$Group
```

Now we plot some of the principal components obtained without centering and scaling.

```r
ggplot(data = pc_no_center_no_scale) +
  geom_point(mapping = aes(x = PC3, y = PC4, colour = Group)) +
  labs(title = "Principal Component Plot",
       x = "Third Principal Component",
       y = "Fourth Principal Component") +
  scale_colour_brewer(palette = "Set1")
```

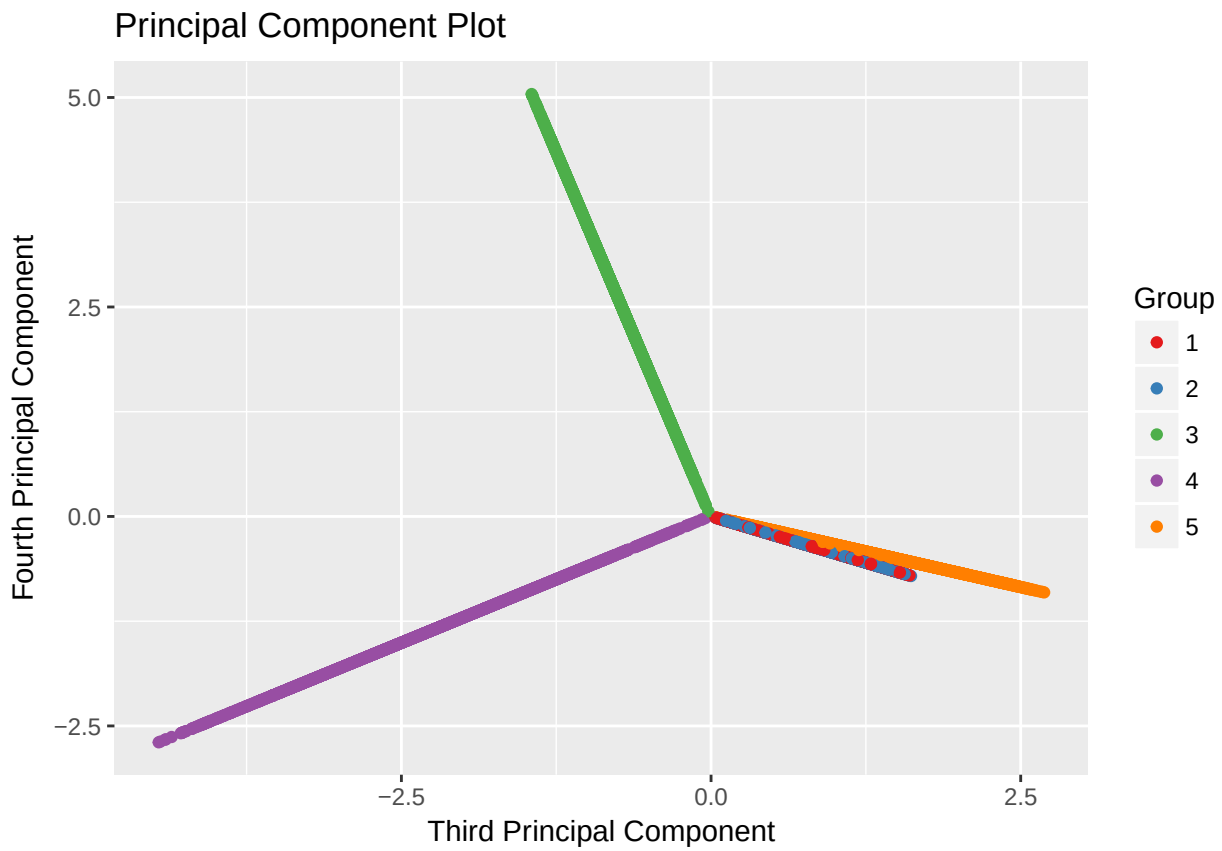



```r
ggplot(data = pc_no_center_no_scale) +
  geom_point(mapping = aes(x = PC1, y = PC2, colour = Group)) +
  labs(title = "Principal Component Plot",
       x = "First Principal Component",
       y = "Second Principal Component") +
  scale_colour_brewer(palette = "Set1")
```

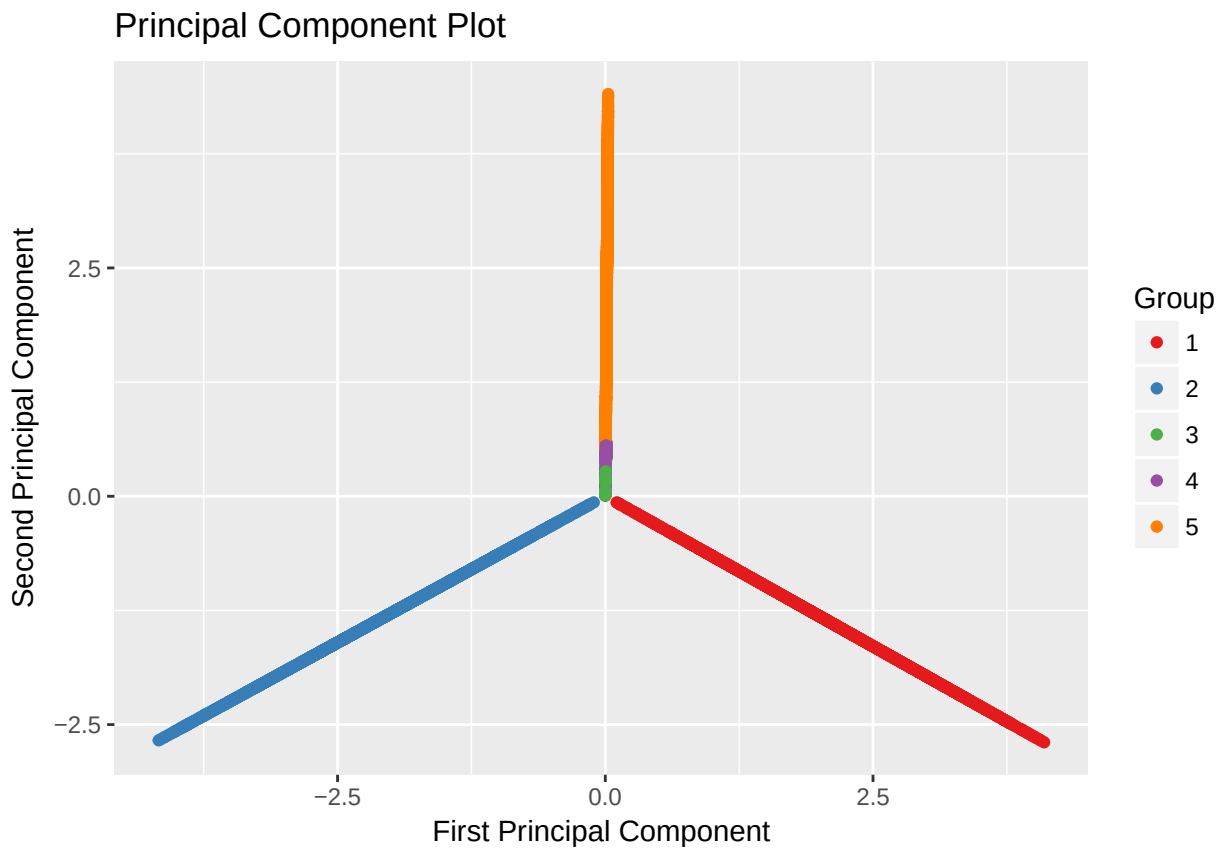


Next we plot some of the principal components obtained by centering and scaling.

```
ggplot(data = pc_center_scale) +
  geom_point(mapping = aes(x = PC3, y = PC4, colour = Group)) +
  labs(title = "Principal Component Plot",
       x = "Third Principal Component",
       y = "Fourth Principal Component") +
  scale_colour_brewer(palette = "Set1")
```




```
ggplot(data = pc_center_scale) +
  geom_point(mapping = aes(x = PC1, y = PC2, colour = Group)) +
  labs(title = "Principal Component Plot",
       x = "First Principal Component",
       y = "Second Principal Component") +
  scale_colour_brewer(palette = "Set1")
```



Both of the principal component plots look identical so centering/scaling didn't seem to make any difference in this case. The point of looking at this was mainly to get an idea of what we'd see performing dimensionality reduction on sparse data with correlated columns. We see that with this type of data all of the correlated data are seen as lines in the principal component plot. Plotting the first and second components against each other we see groups 1, 2, and 5. The information for these groups were encoded in the first and second principal components when performing PCA.

The second plot shows the third and fourth principal components plotted against each other. In that plot we see groups 3 and 4 with groups 1,2, and 5 overlapping with each other. The information for groups 3 and 4 were encoded in the third and fourth principal components.

That's what we get out of PCA.

SVD

Now we perform SVD and see what we get. We conduct SVD both with and without centering/scaling.

```
indices = data.frame(which(data_for_pca != 0,arr.ind = T))
```

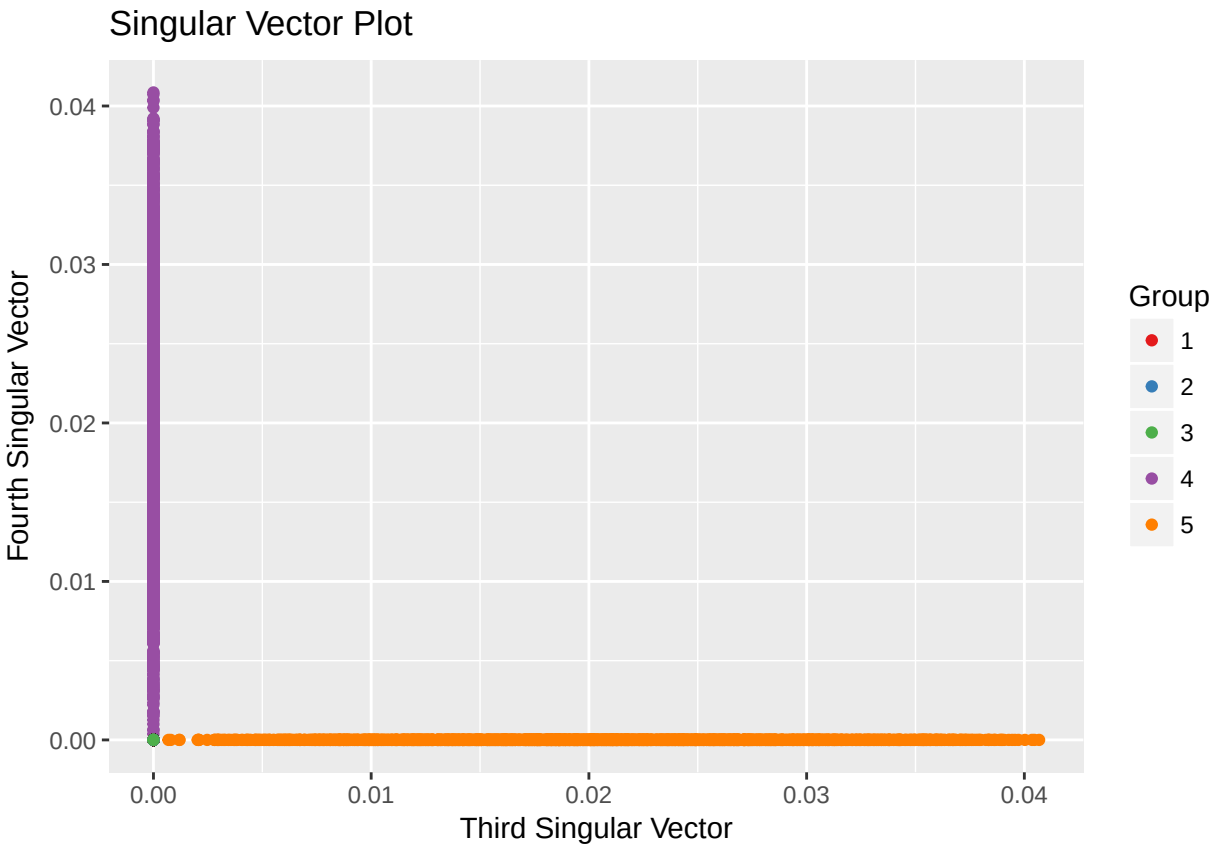
```
## Warning in data.row.names(row.names, rowsi, i): some row.names duplicated:
## 2001,2002,2003,2004,2005,2006,2007,2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018,2019,2020,
```

```
values = c()
for (i in 1:nrow(indices)){
  values[i] = data_for_pca[indices$row[i], indices$col[i]]
}
indices$values = values
```

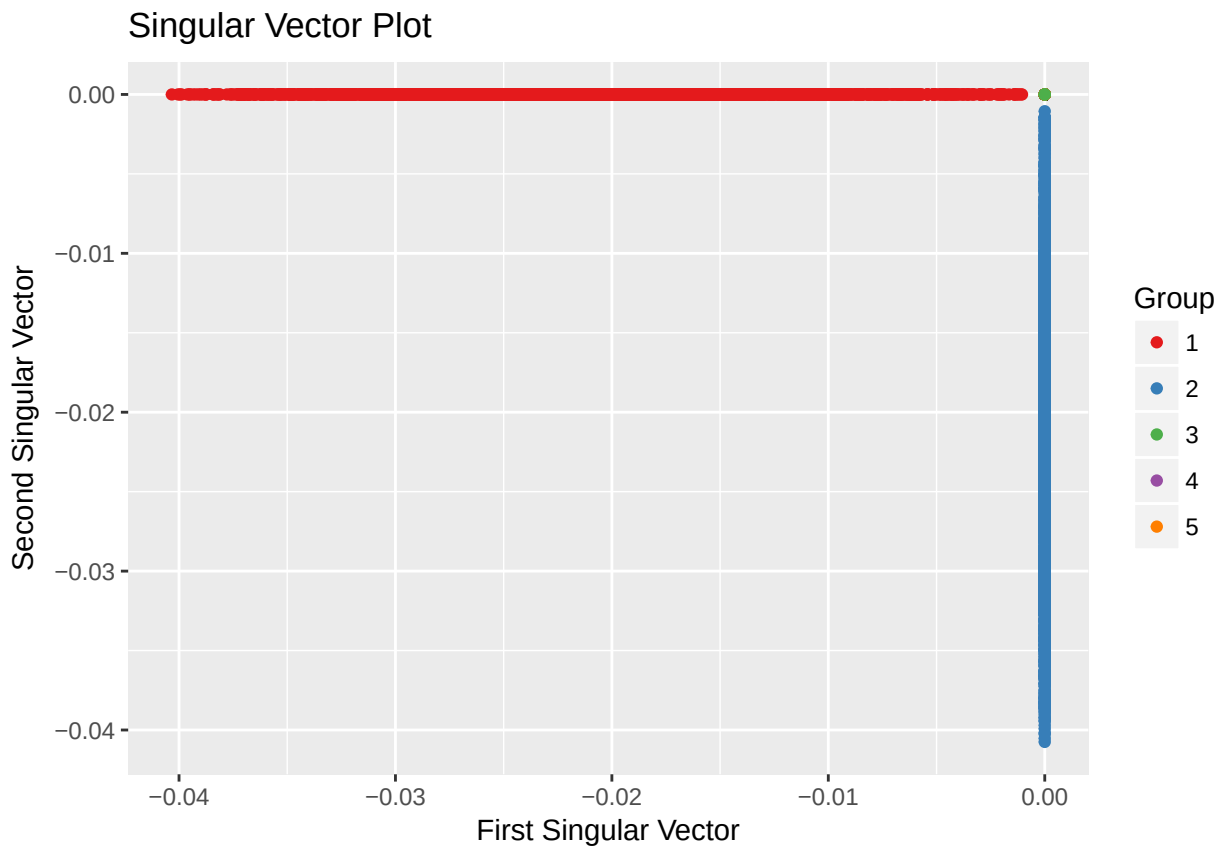
```
matrix = sparseMatrix(i = indices$row, j = indices$col, x = indices$values)
matrix2 = as.matrix(matrix)
column_sdev = colSds(matrix2)
column_means = colMeans(matrix)

svd_irlba_no_center_no_scale = svd(x = data_for_pca,
                                   nu = 5)

svd_irlba_no_center_no_scale = data.frame(svd_irlba_no_center_no_scale$u)
svd_irlba_no_center_no_scale$Group = factor(df$Group)

temp = indices %>%
  group_by(col) %>%
  summarise(mean = mean(values), sd = sd(values))

temp = left_join(indices, temp, by = "col")
temp = temp %>%
  mutate(values_hat = (values - mean)/(sd)) %>%
  select(row, col, values_hat)

colnames(temp)[3] = "values"
matrix = sparseMatrix(i = temp$row, j = temp$col, x = temp$values)


svd_irlba_center_scale = svd(x = matrix,
                             nu = 5)

svd_irlba_center_scale = data.frame(svd_irlba_center_scale$u)
svd_irlba_center_scale$Group = factor(df$Group)
```

First, we plot the results from SVD without centering/scaling.

```
ggplot(data = svd_irlba_no_center_no_scale) +
  geom_point(mapping = aes(x = X3, y = X4, color = Group)) +
  labs(title = "Singular Vector Plot",
       x = "Third Singular Vector",
       y = "Fourth Singular Vector") +
  scale_colour_brewer(palette = "Set1")
```

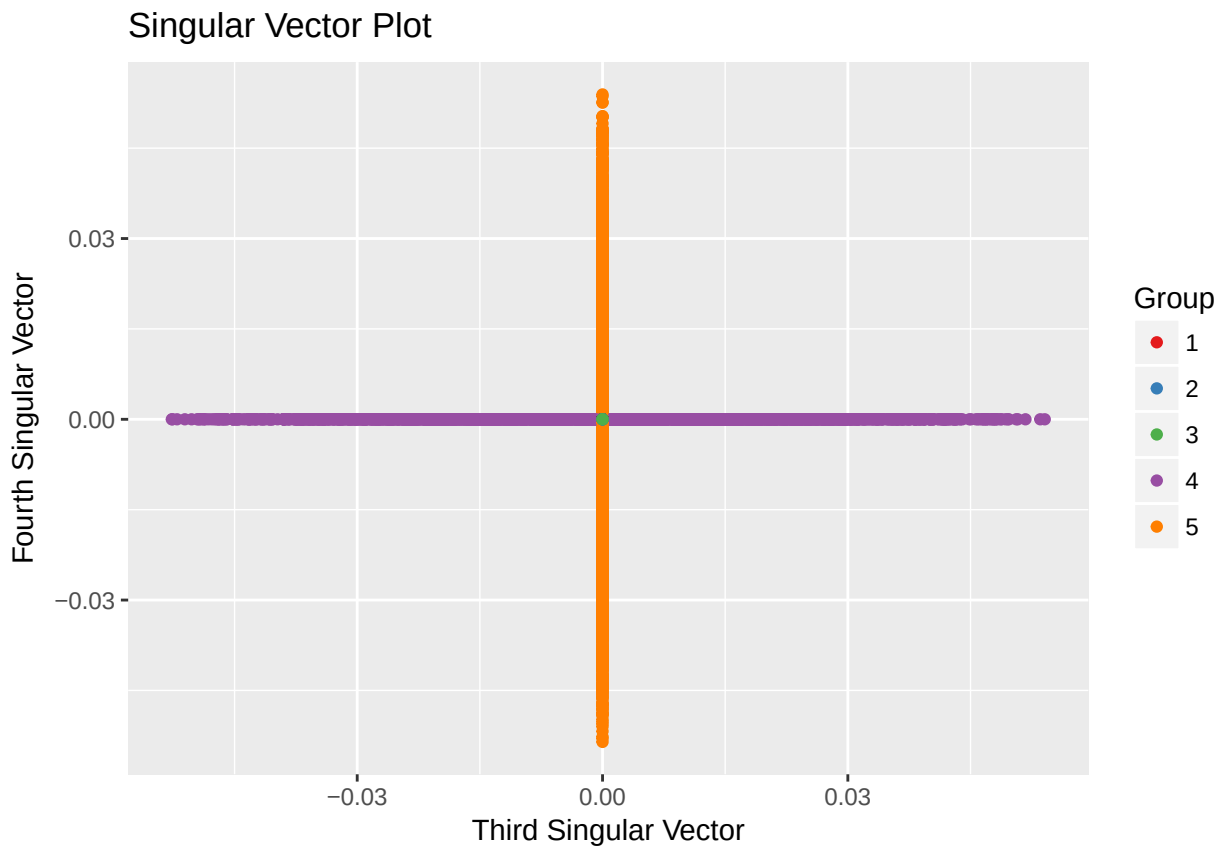



```
ggplot(data = svd_irlba_no_center_no_scale) +
  geom_point(mapping = aes(x = X1, y = X2, color = Group)) +
  labs(title = "Singular Vector Plot",
       x = "First Singular Vector",
       y = "Second Singular Vector") +
  scale_colour_brewer(palette = "Set1")
```

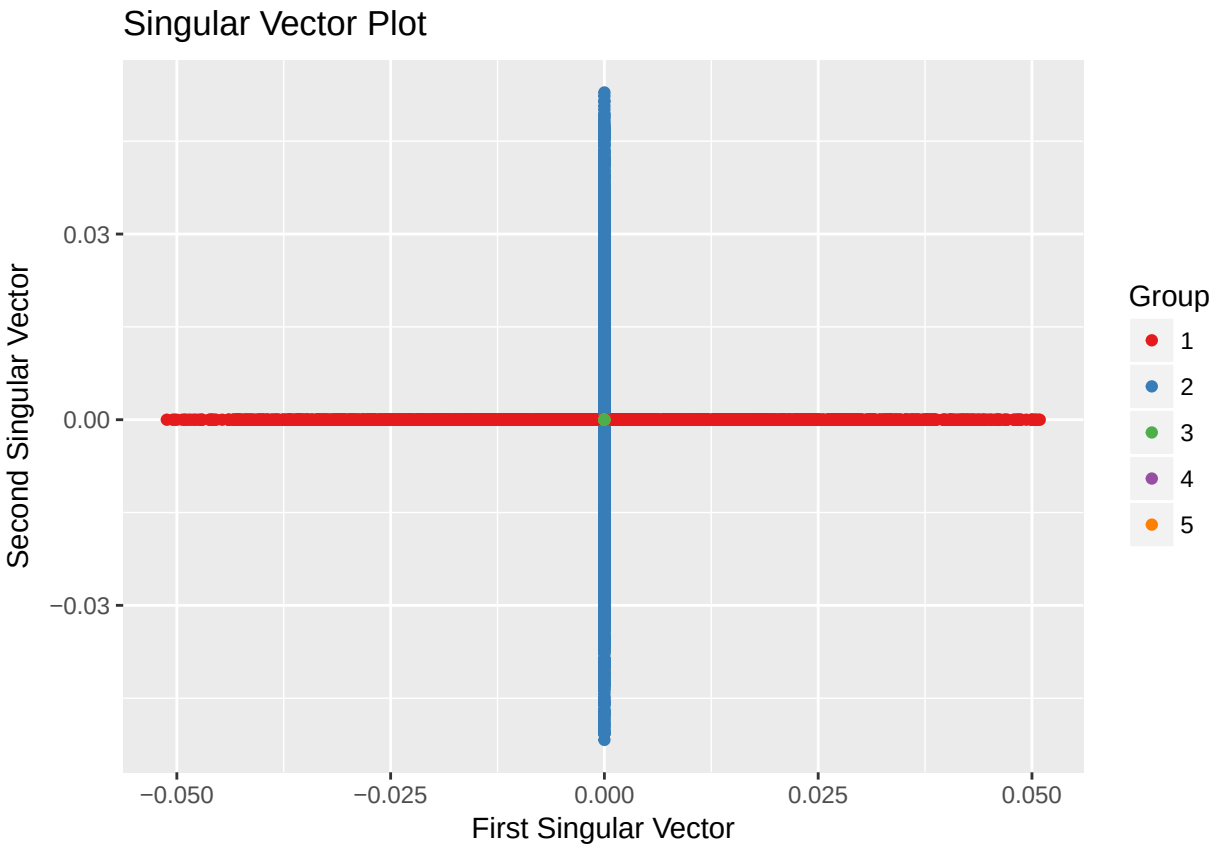


Next we plot the results of SVD with centering/scaling.

```
ggplot(data = svd_irlba_center_scale) +
  geom_point(mapping = aes(x = X3, y = X4, colour = Group)) +
  labs(title = "Singular Vector Plot",
       x = "Third Singular Vector",
       y = "Fourth Singular Vector") +
  scale_colour_brewer(palette = "Set1")
```




```
ggplot(data = svd_irlba_center_scale) +
  geom_point(mapping = aes(x = X1, y = X2, colour = Group)) +
  labs(title = "Singular Vector Plot",
       x = "First Singular Vector",
       y = "Second Singular Vector") +
  scale_colour_brewer(palette = "Set1")
```



The first thing to notice that in this case there did seem to be a difference between SVD with and without centering/scaling. Without centering/scaling the singular vectors appeared as horizontal/vertical lines in the upper right corner of the graph. When we centered/scaled they appeared centered in the middle of the graph.

It appears that the first two groups were encoded in the first and second singular vectors in both of the plots. Plotting the third and fourth singular vectors against each other we see that groups 4 and 5 emerge.

FPDS Data

Now that we've seen PCA/SVD on a synthetically generated data set let's take a look at some real data. Recall that the purpose for looking at PCA/SVD with generated data was to simulate what I might see with the government contracting data I have been working with. There are a couple characteristics that I was looking to reproduce:

1. Sparsity: FPDS data are very sparse; **over 99% of the elements in the matrix are zero!**
2. Correlated columns: the colums in the FPDS data represent NAICS codes which can get quite detailed. There are different NAICS codes for companies involved in aircraft manufacturing and for companies that manufacture screws/bolts, but if a company can build aircraft it probably makes its own screws/bolts too!

We start by performing SVD on the FPDS data. Rows of the data represent companies and columns represent NAICS codes. An element in the table represents the amount of money that was obligated (or de-obligated) to the company for that NAICS code. We look at a few different transformations of this data set:

1. Total dollars obligated or de-obligated. This is the data we just described in the paragraph above.
2. Total dollars obligated/de-obligated capped at the 90th percentile. There are some companies that make a *significant* amount of money and are outliers in the data. Capping at the 90th percentile help

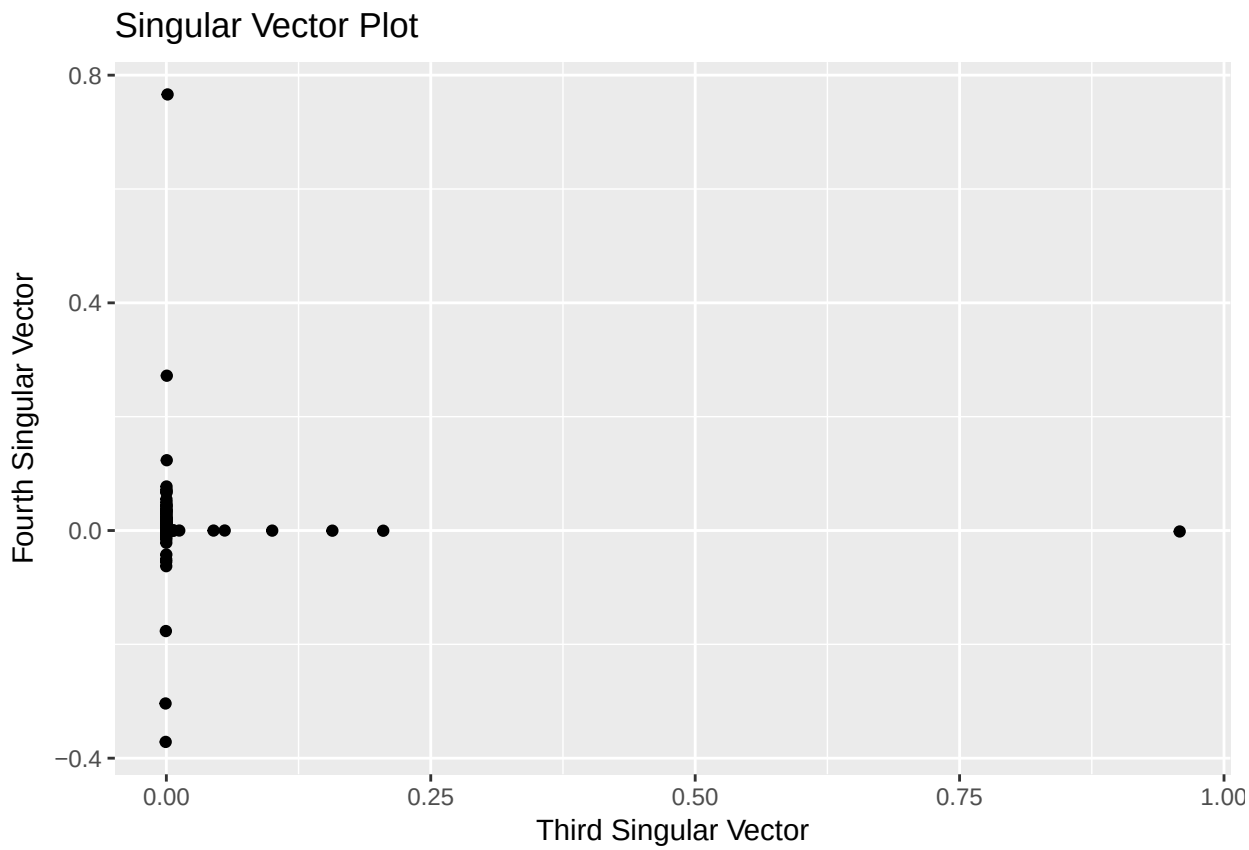
to control for these outliers.

3. We transform rows into percentages. So an element represents the percentage of revenue a company generated under a NAICS code. We also cap percentages from below. Anything that represents less then 1% of revenue is set to 0. Also, since some companies are de-obligated funds (the government takes money back that it promised to spend) we ignore this by taking absolute values of every element assuming that no companies were de-obligated funds.

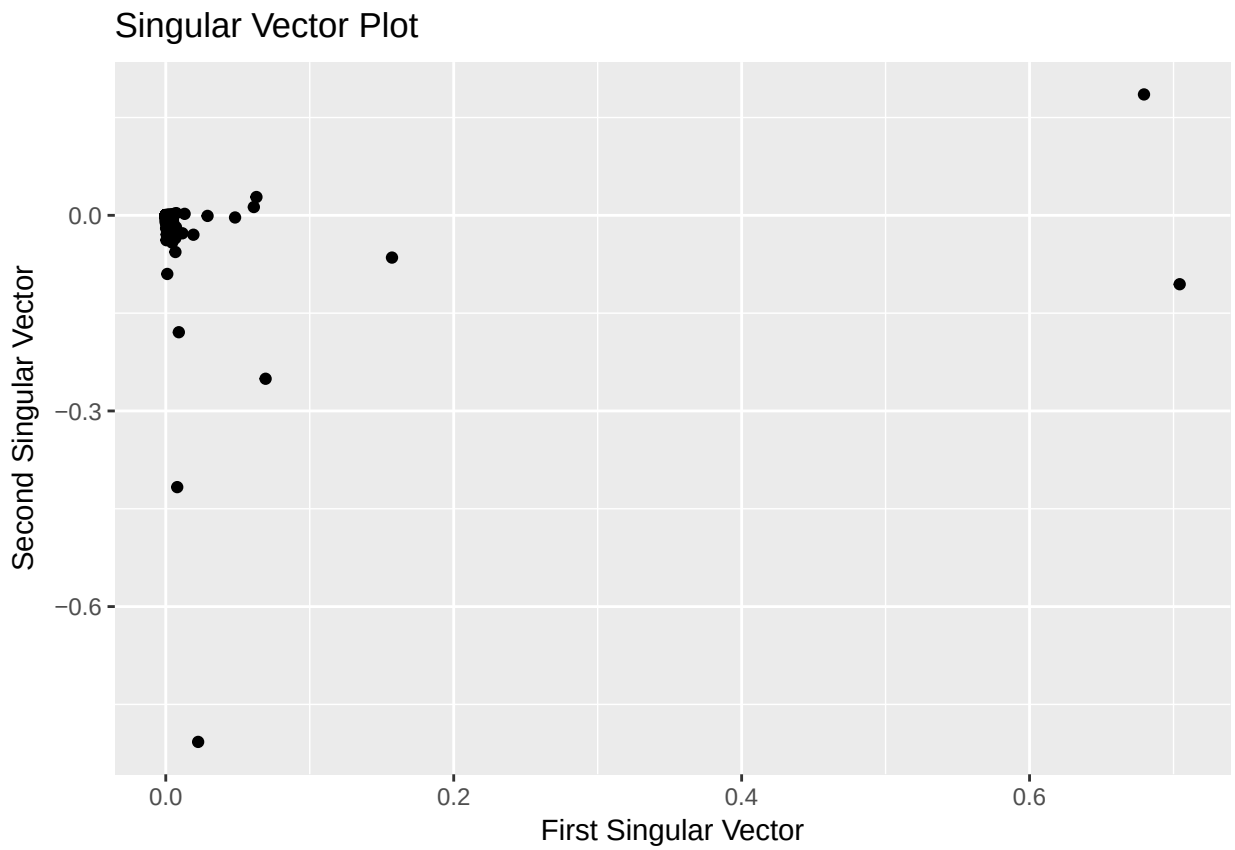
Total Dollars Obligated

```
df = read.csv("data/Total_Dollars_Obligated.csv")
matrix = sparseMatrix(i = df$row, j = df$col, x = df$values)
matrix2 = as.matrix(matrix)
column_sdev = colSds(matrix2)
column_means = colMeans(matrix)

# No center no scale
svd_irlba_no_center_no_scale = irlba(A = matrix,
                                     nu = 5)

svd_irlba_no_center_no_scale = data.frame(svd_irlba_no_center_no_scale$u)
```

```
ggplot(data = svd_irlba_no_center_no_scale) +
  geom_point(mapping = aes(x = X3, y = X4)) +
  labs(title = "Singular Vector Plot",
       x = "Third Singular Vector",
       y = "Fourth Singular Vector")
```




```
ggplot(data = svd_irlba_no_center_no_scale) +
  geom_point(mapping = aes(x = X1, y = X2)) +
  labs(title = "Singular Vector Plot",
       x = "First Singular Vector",
       y = "Second Singular Vector")
```



Most of the points appear on top of each other in the upper left corner. There are some points that appear further out along the horizonal/vertical direction. Very few points appear in the middle of the plot.

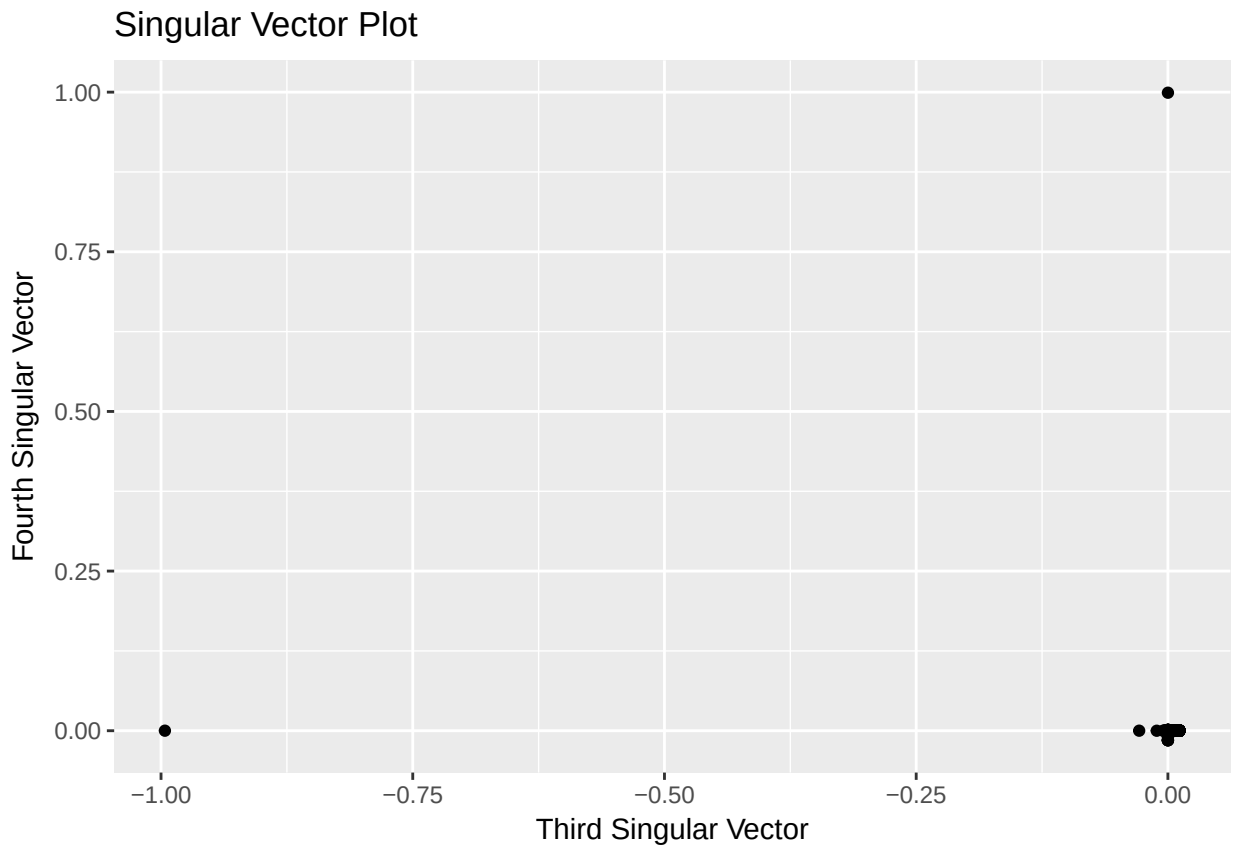
Next we cap the data from above at the 90th percentile.

Total Dollars Obligated Capped

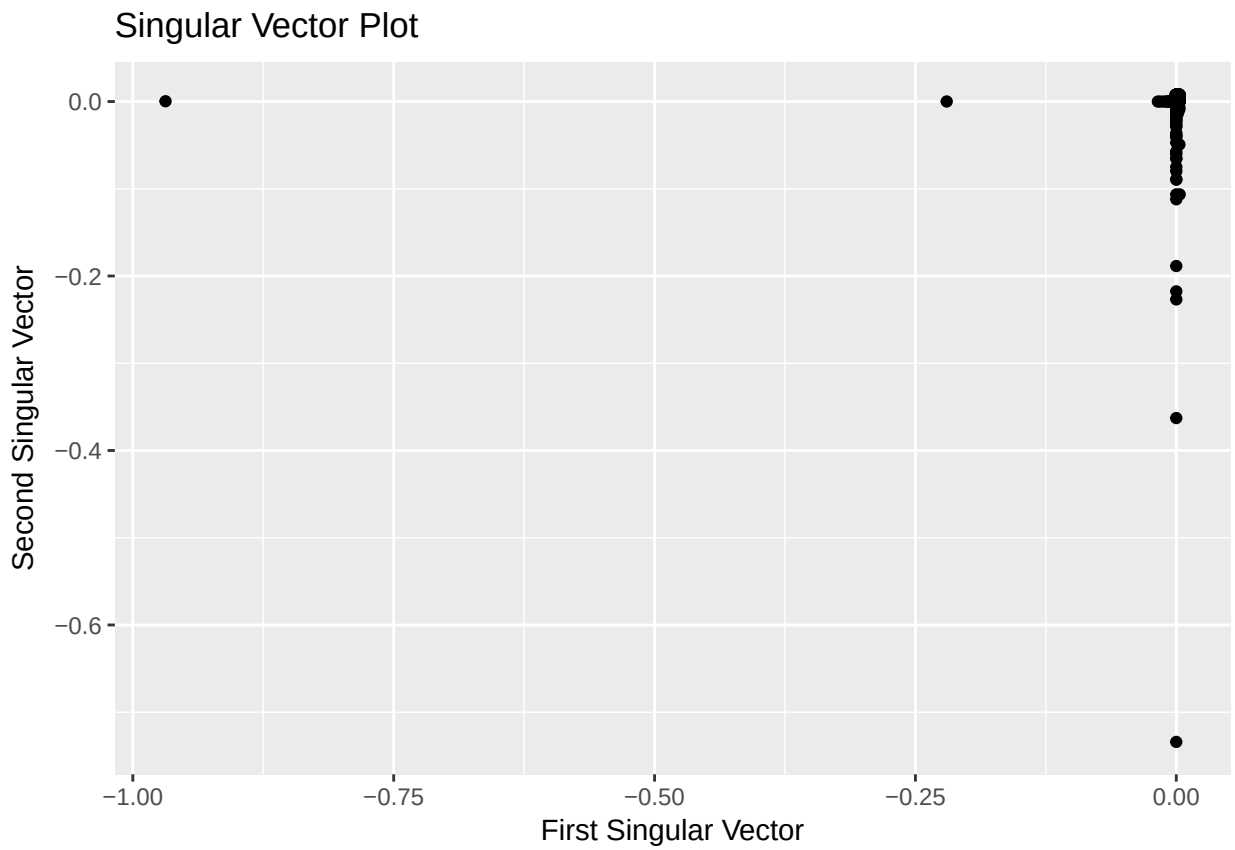
```
df = read.csv("data/Total_Dollars_Obligated.csv")
df = Transform_by_Capping(df = df,
                          lower_bound = 0.01,
                          upper_bound = 0.9,
                          type = "Above")
matrix = sparseMatrix(i = df$row, j = df$col, x = df$values)
matrix2 = as.matrix(matrix)
column_sdev = colSds(matrix2)
column_means = colMeans(matrix)

# No center no scale
svd_irlba_no_center_no_scale = irlba(A = matrix,
                                     nu = 5)

svd_irlba_no_center_no_scale = data.frame(svd_irlba_no_center_no_scale$u)
```

```
ggplot(data = svd_irlba_no_center_no_scale) +
  geom_point(mapping = aes(x = X3, y = X4)) +
```

```
  labs(title = "Singular Vector Plot",
       x = "Third Singular Vector",
       y = "Fourth Singular Vector")
```




```
ggplot(data = svd_irlba_no_center_no_scale) +
  geom_point(mapping = aes(x = X1, y = X2)) +
  labs(title = "Singular Vector Plot",
       x = "First Singular Vector",
       y = "Second Singular Vector")
```



When we cap at the 90th percentile we see that the points along the first and second principal components lie almost exclusively along the horizontal/vertical directions. We see the same thing when we plot the third and fourth singular vector against each other.

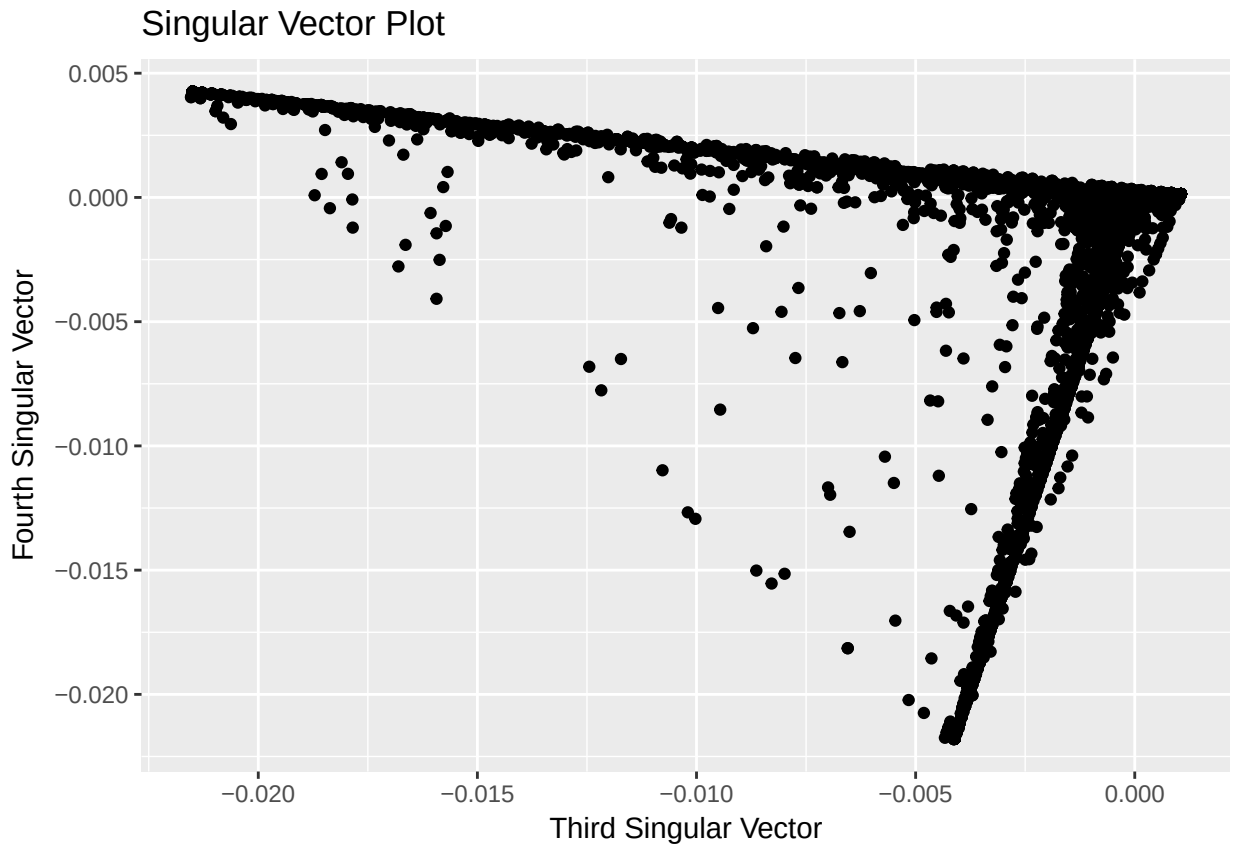
For our last example we transform the data by dividing each value by the sum of the values in its row. Instead of representing total dollars the data are now percentages of revenue a NAICS code makes up for a company.

Total Obligated as Percentage Capped

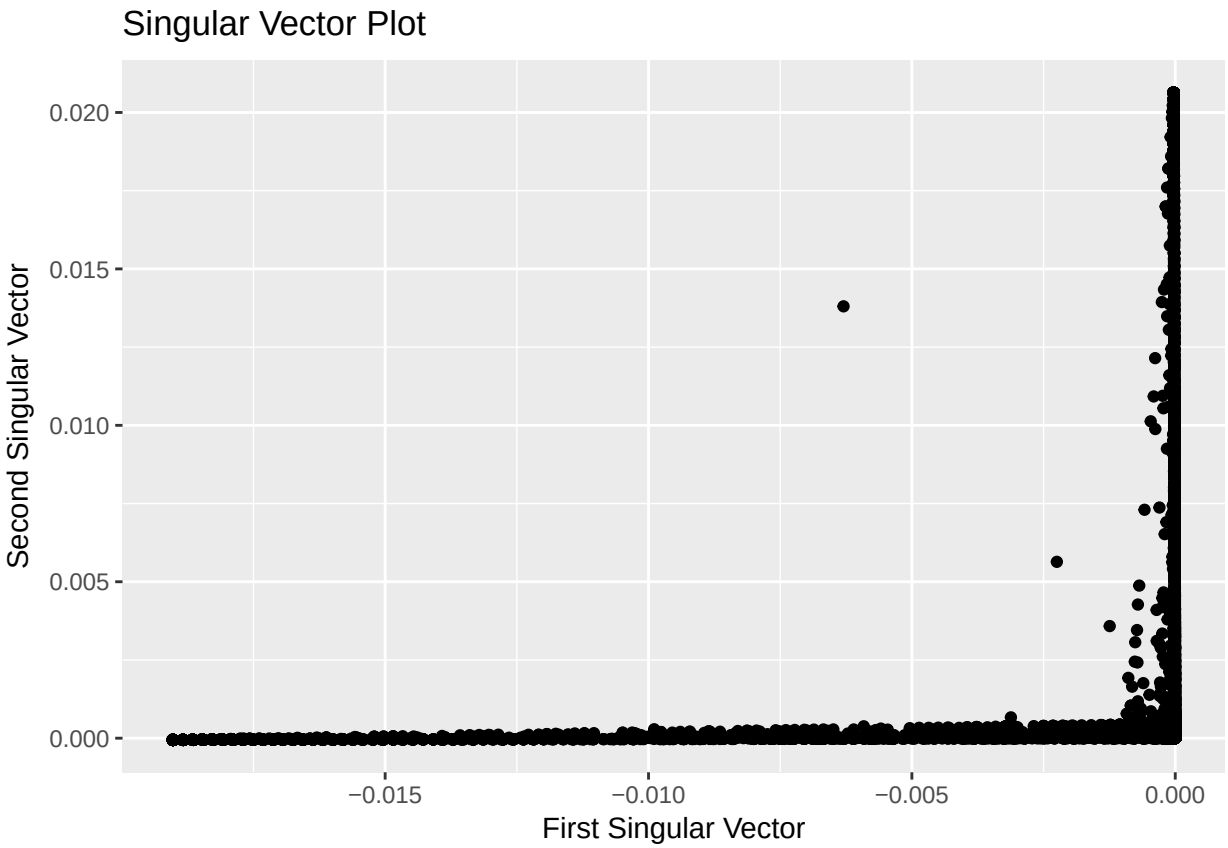
```
df = read.csv("data/Total_Dollars_Obligated.csv")
df = Transform_Negative_Values(df,
                               type = "absolute value")
df = Transform_to_Percentage(df)
df = Transform_by_Capping(df = df,
                          lower_bound = 0.01,
                          upper_bound = 0.9,
                          type = "Below")
matrix = sparseMatrix(i = df$row, j = df$col, x = df$values)
matrix2 = as.matrix(matrix)
column_sdev = colSds(matrix2)
column_means = colMeans(matrix)

# No center no scale
svd_irlba_no_center_no_scale = irlba(A = matrix,
                                     nu = 5)
```

```
svd_irlba_no_center_no_scale = data.frame(svd_irlba_no_center_no_scale$u)
```

```
ggplot(data = svd_irlba_no_center_no_scale) +
  geom_point(mapping = aes(x = X3, y = X4)) +
  labs(title = "Singular Vector Plot",
       x = "Third Singular Vector",
       y = "Fourth Singular Vector")
```




```
ggplot(data = svd_irlba_no_center_no_scale) +
  geom_point(mapping = aes(x = X1, y = X2)) +
  labs(title = "Singular Vector Plot",
       x = "First Singular Vector",
       y = "Second Singular Vector")
```



When we plot the first and second singular vectors against each other we see points branching out in both the horizontal and vertical directions. Very few points appear in the middle. This looks very similar to the SVD plot we saw earlier without centering/scaling on our synthetically generated data. When we plot the third and fourth singular vectors against each other we also see most points appearing in the vertical/horizontal direction, but they are tilted slightly. We also see more points in the middle of the graph.

Conclusion

It seemed that when we transformed rows into a percent the SVD component plots looked much closer to what we saw with our synthetic data. It seems like groups of points did emerge out of the data whereas when we looked at data representing total dollars obligated most points overlapped with each other. We still saw the same behavior with points mainly lying along the horizontal/vertical directions. I also believe that transforming data into percentages was helpful for putting columns on a similar scale. I have tried centering/scaling the data prior to performing SVD, but I get an error. Unfortunately, since the data is sparse it's possible that we might have only one non-zero value in a column. The standard deviation for a single value is undefined so centering and scaling by using the mean and standard deviation is not the right feature scaling approach to use.