PCA ADDENDUM

PSTAT 100 - DATA SCIENCE: CONCEPTS AND ANALYSIS

INSTRUCTOR: Ethan P. Marzban

As mentioned in lecture, **Principal Components Analysis** (PCA) is an incredibly rich and useful topic! Given our limited time, we are only able to scratch the surface in PSTAT 100; you will definitely learn more about PCA in your future PSTAT courses (like PSTAT 131/231). With that said, I figured it may be useful to post this document compiling our (PSTAT 100) treatment of PCA; I also include a bit of extra discussion on topics we didn't get a chance to discuss in Lecture but that I find useful to understand.

Required Packages / Setup

```
library(tidyverse)     ## for plots and wrangling
library(reshape2)      ## for melting data frames
library(ggokabeito)    ## for using the Okabe-Ito color palette
```

Terminology and General Overview

Given a mean-centered data matrix $\mathbf{X}$, we ask ourselves: is it possible to obtain a low-rank approximation (aka *reconstruction*) of $\mathbf{X}$ that preserves as much information as possible? The answer is, "yes!"

Specifically, we first asked ourselves: what is the unit vector $\vec{v}$ for which $\mathbf{X}\vec{v}$, the data projected along $\vec{v}$, has maximum variance over all possible $\vec{v}$? In other words, what is the solution to the following constrained maximization problem:

$$\arg\max_{\vec{v}} \left\{ \vec{v}^{\mathsf{T}} \mathbf{X}^{\mathsf{T}} \mathbf{X} \vec{v} \right\} \quad \text{s.t.} \quad \vec{v}^{\mathsf{T}} \vec{v} = 1 \tag{1}$$

We found that solutions $\vec{v}$ to (1) must satisfy $(\vec{X}^{\mathsf{T}}\mathbf{X})\vec{v} = \lambda\vec{v}$; in other words, the solutions to (1) are precisely the **eigenvectors** of $\mathbf{X}^{\mathsf{T}}\mathbf{X}$.

> **Insight**
>
> Why do we need mean-centered data? Because, we derived (1) by asserting that the variance of $\mathbf{X}\vec{v}$ is simply proportional to
>
> $$\text{Var}(\mathbf{X}\vec{v}) \propto \|\mathbf{X}\vec{v}\|^2 = (\mathbf{X}\vec{v})^{\mathsf{T}}(\mathbf{X}\vec{v}) = \vec{v}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{X}\vec{v}$$
>
> But, this is true only if $\mathbf{X}$ is mean-centered!

The variance of $\mathbf{X}\vec{v}$, again assuming a mean-centered data matrix $\mathbf{X}$, where $\vec{v}$ is a solution to (1) is found to be proportional to λ, the associated eigenvalue. The justification is as follows

$$\text{Var}(\mathbf{X}\vec{v}) \propto= \vec{v}^{\mathsf{T}}(\mathbf{X}^{\mathsf{T}}\mathbf{X}\vec{v}) = \vec{v}^{\mathsf{T}}(\lambda\vec{v}) = \lambda\vec{v}^{\mathsf{T}}\vec{v} = \lambda$$

Insight

The direction $\vec{v}_1$ along which the projected data has *maximum* variance is the eigenvector of $(\mathbf{X}^\mathsf{T}\mathbf{X})$ with *largest* associated eigenvalue; the direction $\vec{v}_2$ along which the projected data has *second largest* variance is the eigenvector of $(\mathbf{X}^\mathsf{T}\mathbf{X})$ with *second largest* associated eigenvalue; etc. The variances of the data projected along these eigenvectors are proportional to the corresponding eigenvalues.

Here is the terminology we are using in PSTAT 100:

- The eigenvectors of $(\mathbf{X}^\mathsf{T}\mathbf{X})$ we call the **principal components**; their elements are called the **principal component lodaings** (or just **loadings**)
- The data projected along the principal components we call the **principal components scores** (or just **scores**)

Caution

The terminology is a bit varied - some people use "PC" to refer to what we are calling the scores.

PCA in Practice

Obtaining the PCs

We saw that there are a couple of different ways to get the PCs:

- Using the $\mathbf{V}$ matrix from the eigendecomposition (EVD) of $(\mathbf{X}^\mathsf{T}\mathbf{X})$
- Using the $\mathbf{V}$ matrix from the singular value decomposition (SVD) of $\mathbf{X}$

As a sample illustration on data:

```
set.seed(100)      ## for reproducibility
x1 <- rnorm(10); x2 <- rnorm(10); x3 <- rnorm(10)
X <- cbind(x1, x2, x3) %>% scale()
```

What this code does is create a (10×3) matrix of full rank by populating each column with a randomly-selected set of numbers. To check that this matrix is indeed full rank:

```
qr(X)$rank
```

```
[1] 3
```

Here are the two ways we can extract out the PCs:

```
(pc_evd <- eigen(t(X) %*% X)$vectors)
```

```
           [,1]        [,2]       [,3]
[1,] -0.5653468  0.62140323  0.5424399
[2,] -0.5117538 -0.77999087  0.3601699
[3,] -0.6469089  0.07397482 -0.7589708
```

```
(pc_svd <- svd(X)$v)
```

```
          [,1]        [,2]       [,3]
[1,] 0.5653468  0.62140323  0.5424399
[2,] 0.5117538 -0.77999087  0.3601699
[3,] 0.6469089  0.07397482 -0.7589708
```

Caution

Recall that, even after fixing them to be of unit norm, eigenvectors are unique only up to a sign flip.

Another way to extract the PCs is using the `prcomp()` function:

```
(pc_prcomp <- prcomp(X, scale. = TRUE)$rotation)
```

```
         PC1         PC2        PC3
x1 0.5653468  0.62140323  0.5424399
x2 0.5117538 -0.77999087  0.3601699
x3 0.6469089  0.07397482 -0.7589708
```

Obtaining the Scores

To find the scores, we (again) have three methods available to us:

- Compute $\mathbf{XV}$
- Compute $\mathbf{U\Sigma}$
- Use `prcomp()`

The equivalence of the first two is established by noting that if $\mathbf{X} = \mathbf{U\Sigma V}^\mathsf{T}$ (i.e. if we begin considering the SVD of $\mathbf{X}$), we have

$$\mathbf{XV} = \mathbf{U\Sigma V}^\mathsf{T}\mathbf{V} = \mathbf{U\Sigma}$$

As an illustration to our same data matrix $\mathbf{X}$ from above (I have used `prcomp()` to extract out the matrix of PCs, but recall that there are other ways to compute this as well!):

```
(scores_xv <- X %*% pc_prcomp)
```

```
              PC1         PC2         PC3
 [1,] -0.86938054 -0.43971072 -0.18111266
 [2,]  0.92448154  0.38798804 -0.91685430
 [3,]  0.05425138  0.37004934 -0.68074026
 [4,]  2.07723422  0.64187409  0.07221006
 [5,] -0.58608073  0.17438623  0.85435816
 [6,] -0.11363220  0.57734188  0.56288248
 [7,] -1.50464040 -0.12468274 -0.14142868
 [8,]  1.24782869  0.59931933  0.41946613
 [9,] -2.48133941  0.03375747 -0.10489104
[10,]  1.25127745 -2.22032292  0.11611009
```

```
(scores_us <- svd(X)$u %*% diag(svd(X)$d))
```

```
             [,1]        [,2]        [,3]
 [1,] -0.86938054 -0.43971072 -0.18111266
 [2,]  0.92448154  0.38798804 -0.91685430
 [3,]  0.05425138  0.37004934 -0.68074026
 [4,]  2.07723422  0.64187409  0.07221006
 [5,] -0.58608073  0.17438623  0.85435816
 [6,] -0.11363220  0.57734188  0.56288248
 [7,] -1.50464040 -0.12468274 -0.14142868
 [8,]  1.24782869  0.59931933  0.41946613
 [9,] -2.48133941  0.03375747 -0.10489104
[10,]  1.25127745 -2.22032292  0.11611009
```

```
(scores_prcomp <- prcomp(X, scale. = TRUE)$x)
```

```
              PC1         PC2         PC3
 [1,] -0.86938054 -0.43971072 -0.18111266
 [2,]  0.92448154  0.38798804 -0.91685430
 [3,]  0.05425138  0.37004934 -0.68074026
 [4,]  2.07723422  0.64187409  0.07221006
 [5,] -0.58608073  0.17438623  0.85435816
 [6,] -0.11363220  0.57734188  0.56288248
 [7,] -1.50464040 -0.12468274 -0.14142868
 [8,]  1.24782869  0.59931933  0.41946613
 [9,] -2.48133941  0.03375747 -0.10489104
[10,]  1.25127745 -2.22032292  0.11611009
```

Obtaining the Variances

Finally, note (again) that there are three ways to obtain the variances of the scores:

- Using the $\mathbf{\Lambda}$ matrix from the EVD of $(\mathbf{X}^\top\mathbf{X})$
- Using $\mathbf{\Sigma}^2$ from the SVD of $\mathbf{X}$
- Using `prcomp()`

As an illustration to our same data matrix $\mathbf{X}$ from above (I have used `prcomp()` to extract out the matrix of PCs, but recall that there are other ways to compute this as well!):

```
(vars_evd <- eigen(t(X) %*% X)$values / (nrow(X) - 1))
```

```
[1] 1.9809441 0.7291396 0.2899164
```

```
(vars_svd <- svd(X)$d^2 / (nrow(X) - 1))
```

```
[1] 1.9809441 0.7291396 0.2899164
```

```
(vars_prcomp <- prcomp(X, scale. = TRUE)$sdev^2)
```

```
[1] 1.9809441 0.7291396 0.2899164
```

If we wanted to, we could verify these empirically by computing the variance of the scores directly - for a large dataset, however, this would not be recommended:

```
(X %*% pc_prcomp) %>% apply(MARGIN = 2, FUN = var)
```

```
      PC1       PC2       PC3
1.9809441 0.7291396 0.2899164
```

Reconstructions

Note that, given that $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$, we have that

$$(\mathbf{X}\mathbf{V})\mathbf{V}^\mathsf{T} = \mathbf{X}$$

Motivated by this, let $\mathbf{V}_d$ denote the matrix whose columns are the first d principal components; then

$$\mathbf{X}_d := \mathbf{X}\mathbf{V}_d\mathbf{V}_d^\mathsf{T}$$

will be a matrix whose dimensions are the same as $\mathbf{X}$ but whose rank will be d. In this way, $\mathbf{X}_d$ can be viewed as a **low-rank approximation** of $\mathbf{X}$.

To illustrate, we can take our data matrix $\mathbf{X}$ from the previous coding examples (which has rank 3) and obtain a two-dimensional (i.e. rank-2) approximation of $\mathbf{X}$ by performing the multiplication

$$\mathbf{X}_2 := \mathbf{X}\begin{bmatrix} | & | \\ \vec{v}_1 & \vec{v}_2 \\ | & | \end{bmatrix}\begin{bmatrix} - & \vec{v}_1^\mathsf{T} & - \\ - & \vec{v}_2^\mathsf{T} & - \end{bmatrix}$$

```
(X_2_direct <- X %*% (pc_prcomp[,1:2]) %*% t(pc_prcomp[,1:2]))
```

```
             x1         x2          x3
 [1,] -0.7647391 -0.1019384 -0.59493753
 [2,]  0.7637497  0.1704798  0.62675668
 [3,]  0.2606207 -0.2608718  0.06247003
 [4,]  1.5732203  0.5623765  1.39126383
 [5,] -0.2229747 -0.4359487 -0.36624065
 [6,]  0.2945205 -0.5084731 -0.03080092
 [7,] -0.9281218 -0.6727540 -0.98258865
 [8,]  1.0778749  0.1711174  0.85156602
 [9,] -1.3818402 -1.2961653 -1.60270335
[10,] -0.6723102  2.3721776  0.64521454
```

```
(X_2_prcomp <- prcomp(X, scale. = T)$x[,1:2] %*% t(pc_prcomp[,1:2]))
```

```
             x1         x2          x3
 [1,] -0.7647391 -0.1019384 -0.59493753
 [2,]  0.7637497  0.1704798  0.62675668
 [3,]  0.2606207 -0.2608718  0.06247003
 [4,]  1.5732203  0.5623765  1.39126383
 [5,] -0.2229747 -0.4359487 -0.36624065
 [6,]  0.2945205 -0.5084731 -0.03080092
```

```
 [7,] -0.9281218 -0.6727540 -0.98258865
 [8,]  1.0778749  0.1711174  0.85156602
 [9,] -1.3818402 -1.2961653 -1.60270335
[10,] -0.6723102  2.3721776  0.64521454
```

Just to check, let's make sure this matrix is of rank 2:

```
qr(X_2_direct)$rank
```

```
[1] 2
```

Insight

The matrix `X2` is a matrix with the same dimensions as the original data matrix $\mathbf{X}$ (namely, 10×3) but with lower rank.

Screeplots

It is, of course, typically impossible to find a low-rank reconstruction of a data matrix that is exactly equal to the original matrix. As such, there is a tradeoff: if we use too low of a rank for our reconstruction, we run the risk of obtaining a reconstruction that is too dissimilar to the original matrix to be of any use. However, if we use too high of a rank (and therefore obtain a very *good* approximation to our original data matrix), we lose the benefit of **dimension reduction** - after all, if we wanted a *perfect* reconstruction of $\mathbf{X}$ with no regard to dimensionality, we would have just picked $\mathbf{X}$ itself!

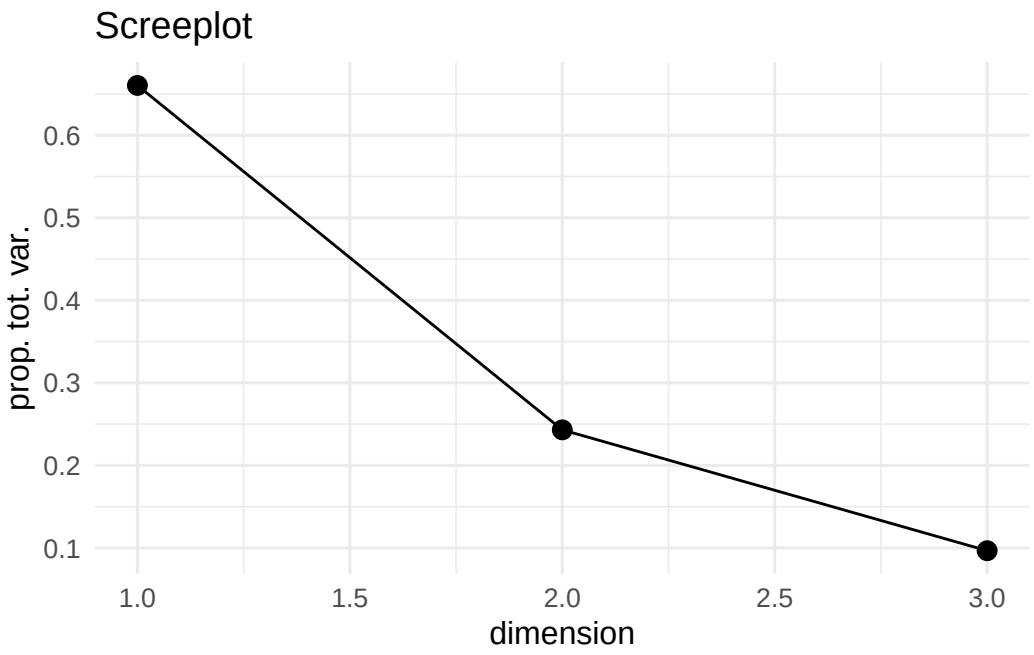
So, a natural question arises: what rank should we use in our reconstruction to (a) get some benefit of dimension reduction, but (b) obtain a reconstruction that is close to our original matrix? The answer is to look at a **screeplot**, which plots the proportion of total *variance* each PC contributes. Specifically, the proportion of total variance explained by the k^{th} PC (i.e. the proportion of total variance that the k^{th} score possesses) is given by

$$s_k = \frac{\lambda_k}{\sum_k \lambda_k} = \frac{\sigma_k^2}{\sum_k \sigma_k^2}$$

where λ_k is the k^{th} eigenvalue of $(\mathbf{X}^\mathsf{T}\mathbf{X})$ and σ_k is the k^{th} singular value of $\mathbf{X}$. A screeplot plots these values on the vertical axis and the dimension (rank) on the horizontal axis.

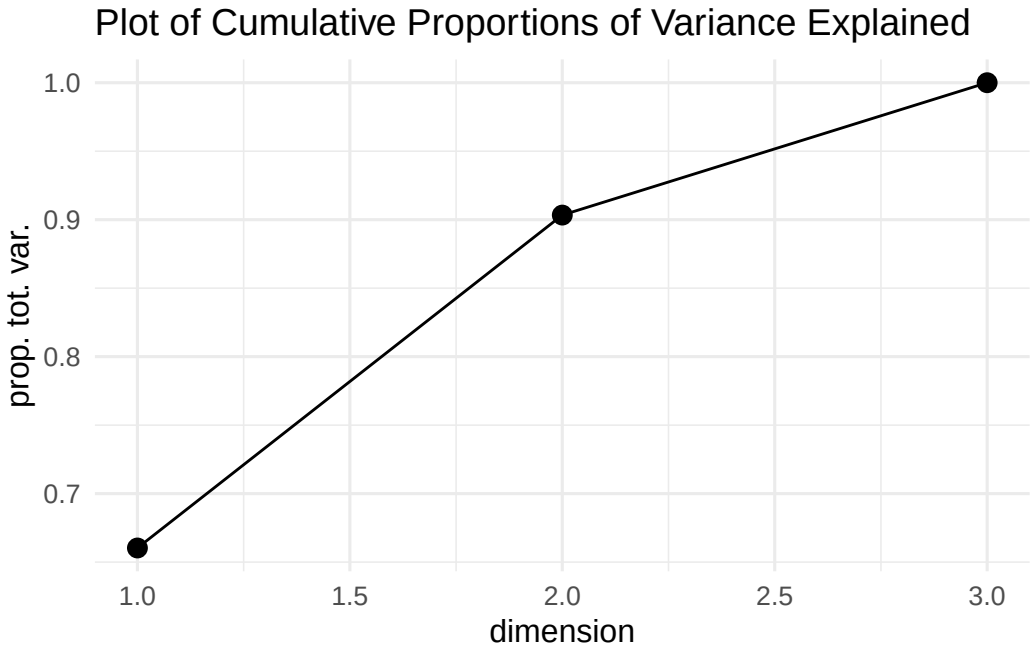
For example, the screeplot associated with our toy data matrix `X` is obtained using

```
s_k <- svd(X)$d^2 / sum(svd(X)$d^2)
data.frame(k = 1:ncol(X), y = s_k) %>% ggplot(aes(x = k, y = s_k)) +
  geom_point(size = 3) + geom_line() + theme_minimal(base_size = 12) +
  xlab("dimension") + ylab("prop. tot. var.") +
  ggtitle("Screeplot")
```



We typically look for an "elbow" in the screeplot, indicating the dimension after which subsequent dimensions contribute neglibly toward the overall variance. Sometimes you will see an alternative to the screeplot, which plots the *cumulative* proportion of variance explained by the *first k* dimensions:

```
cumulative_vars <- (svd(X)$d^2 / sum(svd(X)$d^2)) %>% cumsum()
data.frame(k = 1:ncol(X), y = cumulative_vars) %>% ggplot(aes(x = k, y = cumulative_vars)) +
  geom_point(size = 3) + geom_line() + theme_minimal(base_size = 12) +
  xlab("dimension") + ylab("prop. tot. var.") +
  ggtitle("Plot of Cumulative Proportions of Variance Explained")
```



Mean-Centering vs. Standardizing

The final thing I'd like to mention (and this was something we didn't get a chance to discuss in lecture) is the difference between *mean centering* and **standardizing** a data matrix. Mean-centering a data matrix means shifting each column to have mean zero; standardizing mean-centers and also *rescales* columns to have unit variance.

To illustrate the importance of *standardizing*, let's consider a simple mock dataset comprised of 4 variables where the first three variables have roughly equal variance but the third has a much higher variance:

```
set.seed(10)     ## for reproducibility
y1 <- rnorm(1000); y2 <- rnorm(1000);
y3 <- rnorm(1000); y4 <- rnorm(1000, 0, 10)
Y <- cbind(y1, y2, y3, y4)
Y_mc <- Y %>% scale(, scale = FALSE)    ## mean-centered only
Y_s <- Y %>% scale(, scale = TRUE)      ## standardized
```

What the code above does is create two related matrices: the matrix `Y_mc` is only mean-centered whereas the `Y_s` matrix is *standardized*. To be clear:

```
apply(Y_mc, MARGIN = 2, FUN = var)
```

```
        y1         y2         y3         y4
 0.9837508  1.0899816  1.0348379 107.6798320
```

Now, notice that our $\mathbf{Y}$ matrix is full rank (rank-4) with uncorrelated variables:

```
qr(Y)$rank
```

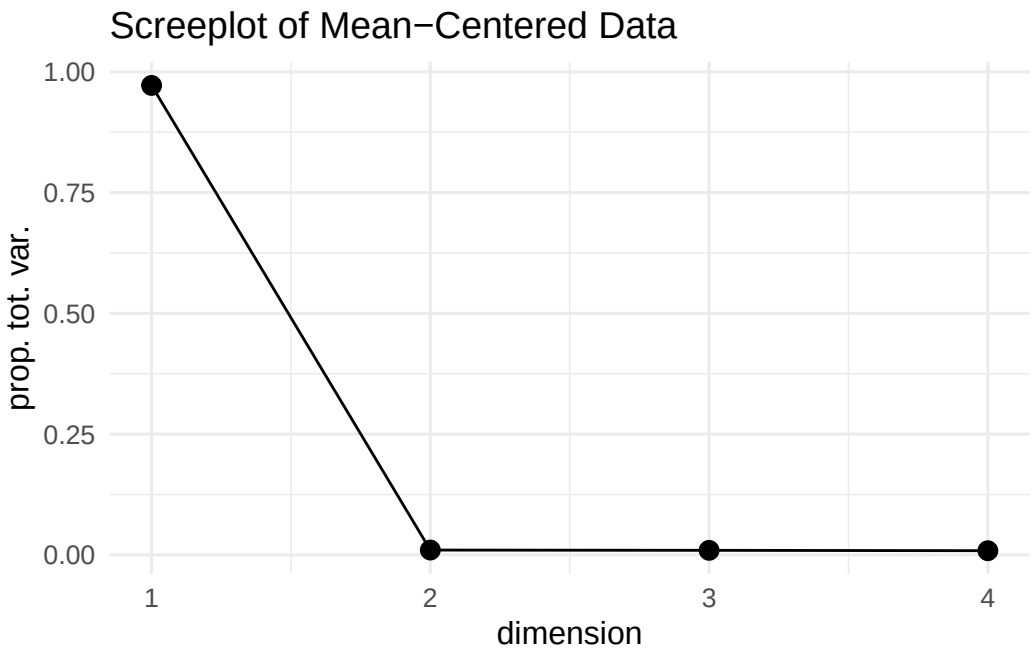
```
[1] 4
```

```
cor(Y)
```

```
             y1           y2          y3           y4
y1  1.000000000  0.044251639 0.030991409 -0.001953145
y2  0.044251639  1.000000000 0.002562131 -0.043828881
y3  0.030991409  0.002562131 1.000000000  0.021962069
y4 -0.001953145 -0.043828881 0.021962069  1.000000000
```

What this means is that, intuitively, the four PCs should contribute roughly equally toward the total variance. The screeplot of the mean-centered data, however, appears to paint a different picture:

```
s_k_mc <- svd(Y_mc)$d^2 / sum(svd(Y_mc)$d^2)
data.frame(k = 1:4, y = s_k_mc) %>% ggplot(aes(x = k, y = s_k_mc)) +
  geom_point(size = 3) + geom_line() +
  xlab("dimension") + ylab("prop. tot. var.") +
  theme_minimal(base_size = 12) + ggtitle("Screeplot of Mean-Centered Data")
```

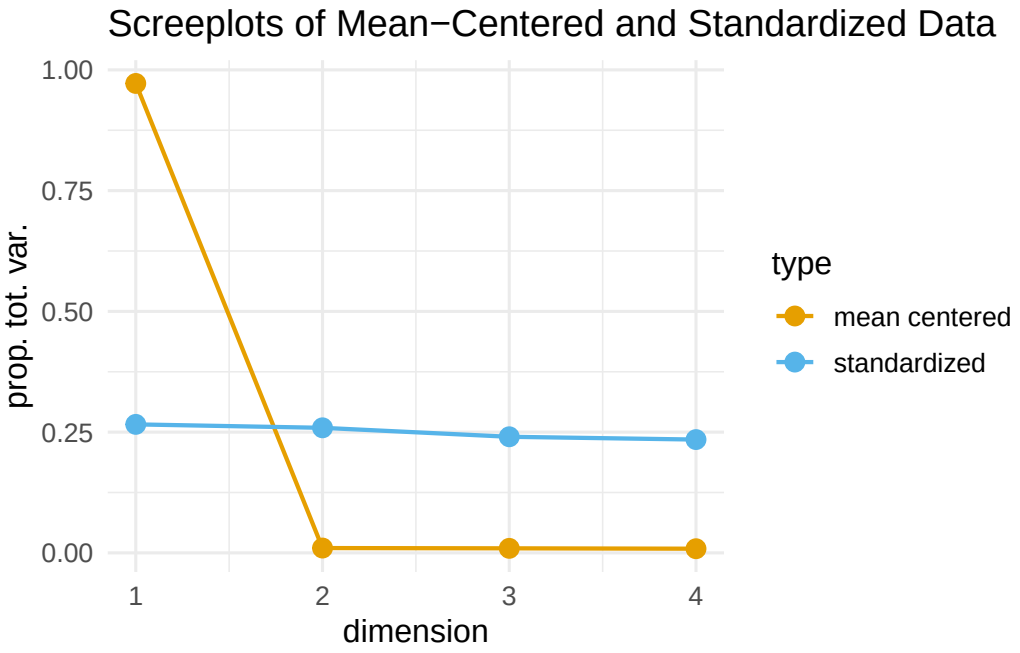


Somehow, the screeplot is identifying the first PC as contributing an enormous amount toward the total variance, when we know for a fact that this shouldn't be the case. That is, the contributions of the first PC are being *drastically* overstated.

The reason this is happening is because one of our variables has a much higher variance than the others! If we take a look at the screeplot of *standardized* data in comparison to that of the *mean-centered* data, we see a much better picture:

```
s_k_s <- svd(Y_s)$d^2 / sum(svd(Y_s)$d^2)

data.frame(x = 1:4,
           `mean centered` = s_k_mc,
           `standardized` = s_k_s,
           check.names = F) %>%
  melt(id.vars = 'x',
       variable.name = "type") %>%
  ggplot(aes(x = x, y = value)) +
  geom_point(aes(colour = type), size = 3) +
  geom_line(aes(colour = type), linewidth = 0.75) +
  xlab("dimension") + ylab("prop. tot. var.") +
  theme_minimal(base_size = 12) +
  ggtitle("Screeplots of Mean-Centered and Standardized Data") +
  scale_color_okabe_ito()
```



Insight

In PCA, certain variables may be overinflated in terms of influence based solely on the fact that they have large variances. As such, it is a good idea to *standardize* our data before performing PCA - not just mean-center it.